DAVID DA ASSUNÇÃO VALLIM

EXTRAÇÃO DE REGRAS DE INTEGRIDADE REFERENCIAL A PARTIR DO CÓDIGO FONTE

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Informática, Curso de Pós-Graduação em Informática, Departamento de Informática, Universidade Federal do Paraná.

Orientador: Prof. Dr. Marcos Sfair Sunye

CURITIBA
2002

Ministério da Educação
Universidade Federal do Paraná
Mestrado em Informática

PARECER

Nós, abaixo assinados, membros da Banca Examinadora da defesa de Dissertação de Mestrado em Informática, do aluno *David da Assunção Vallim*, avaliamos o trabalho intitulado, ***"Extração de Regras da Integridade Referencial a partir do Código Fonte"***, cuja defesa foi realizada no dia 17 de setembro de 2002, às quatorze horas, no anfiteatro A do Setor de Ciências Exatas da Universidade Federal do Paraná. Após a avaliação, decidimos pela aprovação do candidato.

Curitiba, 17 de setembro de 2002.

Prof. Dr. Marcos Sfair Sunyé
DINF/UFPR - Orientador

Prof. Dr. Ricardo Pereira e Silva
UFSC - Membro Externo

Prof. Dr. Carlos Carvalho
DPTº FÍSICA/UFPR

Profª. Dra. Laura Sanchez Garcia
DINF/UFPR

MINISTÉRIO DA EDUCAÇÃO - UNIVERSIDADE FEDERAL DO PARANÁ - Setor de Ciências Exatas - Coordenação em Informática

AGRADECIMENTOS

A Deus
Porque a Ele tudo devemos e a Ele sou grato.

Ao meu orientador
Por ter me aceito como seu orientando, pela motivação e esclarecimentos indispensáveis para a conclusão deste trabalho.

Aos meus amigos
Pelo incentivo para que eu não esmorecesse e levasse essa empreitada até o fim.

SUMÁRIO

LISTA DE TABELAS vi
RESUMO ix
ABSTRACT x
1. INTRODUÇÃO 1
1.1. A MOTIVAÇÃO 1
1.2. O CONTEXTO 2
1.3. O OBJETIVO 5
2. REGRAS DE INTEGRIDADE 7
2.1. ENTIDADES 7
2.2. ATRIBUTOS 7
2.3. RELAÇÕES 8
2.4. CONECTIVIDADE E CARDINALIDADE 8
2.5. CHAVE PRIMÁRIA E CHAVE ESTRANGEIRA 9
2.6. INTEGRIDADE NO MODELO RELACIONAL 9
2.6.1. Integridade de Dados 9
2.6.2. Integridade de Entidade 9
2.6.3. Integridade Referencial 9
2.6.4. Regras de Inserção 10
2.6.5. Regra de Exclusão 10
2.7. APLICAÇÃO E UTILIZAÇÃO DO MODELO 10
3. ENGENHARIA REVERSA 12
3.1. DESAFIOS PARA ENGENHARIA REVERSA 13
3.2. A EVOLUÇÃO DA ENGENHARIA REVERSA 15
3.3. FERRAMENTAS DE ENGENHARIA REVERSA 17
3.4. ESTADO DA ARTE 17
3.5. COMPARANDO FERRAMENTAS DE ENGENHARIA REVERSA 18
3.6. AVALIAÇÃO DE FERRAMENTAS DE ENGENHARIA REVERSA 20
3.7. OBTENÇÃO DE REGRAS DE RESTRIÇÃO 21
3.8. CONSIDERAÇÕES 22
4. PROPOSTA 24

4.1. ESCOLHA DO SISTEMA 25
4.2. CARACTERÍSTICAS DO SISTEMA 25
4.3. IDENTIFICAÇÃO DOS PADRÕES EXISTENTES 26
4.4. PADRÕES DA APLICAÇÃO 27
4.5. VERIFICAÇÃO DAS REGRAS DE INTEGRIDADE 30
4.6. EXTRAÇÃO DE INFORMAÇÕES 31
4.7. VERIFICAÇÃO DE INTEGRIDADE REFERENCIAL 32
4.8. EXTRAINDO INFORMAÇÕES 33
4.9. ARMAZENAMENTO DA INFORMAÇÕES 33
5 RESULTADO 36
6 CONCLUSÃO 39
6.1 QUANTO AOS OBJETIVOS 39
6.2 QUANTO AOS RESULTADOS 39
6.3 TRABALHOS FUTUROS 40
REFERÊNCIAS 41
ANEXOS 44

LISTA DE QUADROS

QAUDRO 1 - TABELA COMPARATIVA DAS FERRAMENTAS.....19
QUADRO 2 – SIGLA DE IDENTIFICAÇÃO DE MÓDULOS.....27
QUADRO 3 – PADRÃO PARA IDENTIFICAÇÃO DE TABELAS SQL.....27
QAUDRO 4 – PADRÃO PARA IDENTIFICAÇÃO DE APLICAÇÕES.....27
QUADRO 5 - PADRÃO PARA IDENTIFICAÇÃO DE
COMPONENTES DA APLICAÇÃO.....27

LISTA DE FIGURAS

FIGURA 1 – EXEMPLO DE RELAÇÃO DE INTEGRIDADE.....26
FIGURA 2 – APRESENTAÇÃO DA APLICAÇÃO NO VISUALAGE.....28
FIGURA 3 – MODELEO CONCEITUAL PARA ARMAZENAMENTO 35

LISTA DE SIGLAS

HC-UFPR..... HOSPITAL DE CLÍNICAS DA UNIVERSIDADE FEDERAL DO PARANÁ

SUS ..... SISTEMA ÚNICO DE SAÚDE

SGBD..... SISTEMA GERENCIADOR DE BASE DE DADOS

SQL..... STRUCTURED QUERY LANGUAGE

SIH-HC ..... SISTEMA DE INFORMAÇÕES HOSPITALARES DO HOSPITAL DE *CLÍNICAS*

UFPR ..... UNIVERSIDADE FEDERAL DO PARANÁ

SQL/DS..... STRUCTURED QUERY LANGUAGE / DATA SERVICE

CSP ..... CROSS STRUCTRED PROGRAM

IBM..... INTERNATIONAL BUSINESS MACHINE

DB2..... DATA BASE 2

ER ..... ENTIDADE RELACIONAL

CASE ..... COMPUTER-AIDED SOFTWARE ENGINEERING

OO..... ORIENTAÇÃO A OBJETOS

ANSI/IEEE .. AMERICAN NATIONAL STANDARDS INSTITUTE/ INSTITUTE OF ELECTRICAL AND ELECTRONIC ENGINEERS

WEB..... WORLD WIDE WEB

COBOL..... COMUM BUSINESS ORIENTED LANGUAGE

API..... APPLICATION PROGRAM INTERFACE

EC ..... EXAMES COMPLEMENTARES

SADT..... SERVIÇO DE APOIO DIAGNÓSTICO E TERAPÊUTICOS

ESF ..... EXTERNAL SOURCE FORMAT

NRF..... NOT RECORD FOUND

RESUMO

Para atender a solicitações de manutenção em sistemas de informação, os responsáveis pelo sistema necessitam conhecer suas funcionalidades, regras de negócio, ou pelo menos, a parte do sistema que será afetada pela alteração. Esta pode ser uma tarefa simples quando executada por quem construiu o sistema, mas se torna complexa quando o projetista original foi deslocado para outras funções ou não está mais na empresa e não existe documentação para obter as informações necessárias. Neste caso, para ajudar a resolver o problema é necessário buscar informações em todos os lugares onde possam existir. Elas podem ser obtidas do esquema do banco de dados, com colegas que participaram na construção do sistema, com o usuário e nos códigos fonte dos programas, sendo este último um dos recursos mais utilizados. Quando se pretende obter informações gerenciais com cruzamento de variáveis, a falta do conhecimento das relações e regras de integridade, podem levar a resultados imprecisos ou equivocados e comprometer decisões importantes. Para fazer reengenharia de sistemas ou para criação de bases de informações gerencias o conhecimento de relações e regras de integridade são de igual forma fundamental. Por isso a proposta deste trabalho é mostrar que é possível identificar, extrair e armazenar regras de integridade referencial do código fonte para subsidiar estas atividades.

Palavras-Chave: Engenharia Reversa; Regras de Integridade; Integridade Referencial.

ABSTRACT

In order to attend the information system maintenance demands an information technology professional needs to know the system as well as its functionalities and business norms, or at least the system portion that will be affected by the alteration. This can be a simple task when performed by the one who developed the system, but it can become a complex one when the system original developer has been transferred to other activities or is no longer working for the company and there's not enough documentation to obtain the needed information. In this case, to help solving the problem, it's necessary to search for information everywhere it may be available. The information is often searched for in the data base schema, with other professionals who took place in the system developing process, with the system users and in the written programs, being this one of the most used ones. To extract relational semantics from source code program is not an easy nor a fast way of obtaining the needed information and this is a proposal for this task.

Key-words: Reverse Engineering, Relational Semantics, Referential Constraints.

1. INTRODUÇÃO

1.1. A MOTIVAÇÃO

Para atender a uma solicitação de manutenção ou melhorias de um sistema, o profissional de informática precisa conhecer cada módulo, seus componentes e funções, para poder implementar o que está sendo solicitado de forma rápida e segura. É necessário que se conheça todas as relações e regras de integridade envolvidas para não introduzir erros no sistema.

Por ser o Hospital de Clínicas da Universidade do Paraná (HC-UFPR) um prestador de serviços para o Sistema Único de Saúde (SUS), subordinado ao Ministério da Saúde, precisa constantemente adequar o seu sistema de informações hospitalares para atender a portarias e normalizações editadas por estes órgãos.

Quando há pressa no desenvolvimento ou alteração de sistemas os projetistas procuram alternativas para simplificar a tarefa. Nestas ocasiões a documentação deixa de ser feita e as decisões de projeto passam a ser feitas de forma verbal e sem nenhum registro.

Dificuldades na implementação e testes das aplicações levaram os projetistas iniciais do SIH-HC a decidir que a integridade referencial entre tabelas, detalhada no capítulo 2, fosse feita pelas aplicações e não pelo SGBD.

Algumas vezes são necessárias informações para as quais não foram previstas funções para recuperá-las no sistema. Como não existe modo de prever todos os dados e relatórios que os usuários necessitam a cada momento para tomadas de decisão ou para pesquisa, utilizam-se rotinas com comandos SQL [SQL/DS], escritas em cada situação, para obter as informações desejadas. Nestes casos o conhecimento das relações entre as tabelas e os atributos envolvidos é fundamental.

Com a troca ou saída do projetista original e a falta de documentação completa e atualizada, este profissional depende em grande parte do código fonte das aplicações, do usuário, da documentação escassa e de consulta a colegas que possam ter alguma

informação para buscar o conhecimento que os projetistas originais tinham e não está documentado.

Qualquer que seja a solicitação do usuário, ela precisa ser atendida rapidamente, independente do motivo, porque a demora pode afetar o atendimento de pacientes, comprometer uma decisão gerencial ou o faturamento dos serviços prestados.

1.2. O CONTEXTO

O Hospital de Clínicas da Universidade Federal do Paraná (HC-UFPR) é um hospital universitário de grande porte, com mais de 600 leitos, 236 ambulatórios, 3.433 funcionários, mensalmente são realizados em média 1.700 internações, 60.000 atendimentos ambulatoriais e 80.000 exames complementares e 742 cirurgias.

O Sistema de Informações Hospitalares do Hospital de Clínicas (SIH-HC) da UFPR, começou a ser desenvolvido em 1990 e conta atualmente com dezesseis módulos implantados, com cerca de 700 tabelas e cerca de 3000 aplicações para manter e dar suporte aos serviços de atendimento a pacientes e serviços administrativos.

Foi originalmente desenvolvido utilizando-se a linguagem CSP - *Cross Sysytem Product* [CSP] e posteriormente substituído pelo *VisualAge Generator* [VAGen], ambos da IBM. O SIH-HC é executado num computador IBM S/390, com IBM/DB2 [IBM/DB2], para atender uma estrutura de 600 pontos de rede, com cerca de 250 pontos ativos simultâneos, funcionando em terminais de modo texto modelo IBM 3270 e microcomputadores com emulação de terminal, operando 24 horas por dia e 365 dias por ano.

O desenvolvimento do SIH-HC da UFPR não foi diferente do que acontece na maioria das organizações em relação à pressão sobre custo, tempo de desenvolvimento, rotatividade de pessoal, falta de documentação e uma longa fila de espera para desenvolvimento de novas funcionalidades.

Ao longo dos anos todos os módulos sofreram modificações, para acrescentar e modificar funcionalidades, e a maioria destas modificações não foi documentada.

Alguns usuários assumiram outras funções dentro da instituição, analistas foram alocados para outros sistemas ou saíram da instituição e alguns módulos já sofreram tantas modificações que cada vez fica mais difícil e demorado fazer novas modificações. Na medida que os novos módulos foram sendo implantados, o número de usuários aumentou e o número de analistas e programadores diminuiu. Além disso, a demanda por novos módulos e funcionalidades aumenta a cada dia.

Cada módulo do SIH-HC atende e dá suporte a serviços específicos de atendimento a pacientes, como por exemplo: Módulo de Identificação Hospitalar, que na verdade trata do cadastro de pacientes; Módulo de Atendimento Ambulatorial, que atende funções de gerenciamento e controle de atendimentos ambulatoriais; Módulo de Exames Laboratoriais, que atende funções de gerenciamento e controle de requisições e registro de resultados de exames laboratoriais; Módulo de Internação, para gerenciamento e controle de pacientes internados; Módulo de Farmácia, para atender funções de prescrição médica e controle de medicamentos; Módulo de Exames Complementares, que trata funções de gerenciamento e controle de requisições, agendamento e laudos de exames complementares (Raios-X, endoscopia, mamografia, neurologia e outros).

Uma característica importante do SIH-HC é de ter um esquema único, projetado para restringir a redundância de dados, fazendo com que as regras de integridade referencial da base de dados seja uma das características importantes do sistema. No capítulo 2, a seguir, estão colocados os principais aspectos em relação à integridade referencial de que trata este trabalho.

Quando um administrador necessita de informações gerenciais para tomada de decisão precisa recorrer a dados consolidados e análise de indicadores tais como produção de cada unidade, seu faturamento, custos por procedimento, taxa de ocupação de leitos, taxa de mortalidade, atendimentos efetuados por origem (cidade, estado, país) do paciente, atendimentos por serviço ou por profissional, dentre muitos outros. Uma informação gerencial particularmente relevante refere-se ao custo de

operação de cada atividade e sua análise em função das inúmeras variáveis envolvidas em cada uma destas atividades.

Um aspecto importante das informações gerenciais é seu caráter variável e muitas vezes imprevisível, pois o diagnóstico de problemas e identificação de suas causas exige freqüentemente a obtenção de novos indicadores e/ou novas correlações de variáveis. Eventualmente, necessita fazer o cruzamento de dados pertencentes a módulos diferentes para obtenção e análise de novos indicadores. Essas características impedem que o uso de informações seja limitado a um conjunto fixo de relatórios.

O SIH-HC possui uma base de dados bastante rica, cobrindo dez anos de funcionamento da instituição. Todavia, esses dados são gerados por um sistema orientado aos procedimentos operacionais do hospital. Essa situação dificulta bastante a disponibilização das informações gerenciais necessárias à administração da instituição e não satisfaz aos requisitos de um sistema gerencial

A necessidade dessas informações tem aumentado e deve continuar a crescer ao longo do tempo por diversas razões, entre as quais podemos citar:

a) Escassez de recursos, obrigando ao aumento da eficiência e diminuição de custos;
b) Redução contínua do quadro de pessoal devido a restrições de contratação;
c) Aumento da complexidade do processo de gerenciamento dos hospitais devido à sofisticação de procedimentos, e
d) Exigência cada vez maior de qualidade de atendimento por parte do Ministério da Saúde.

Os dados do SIH-HC também estão disponíveis para pesquisadores e estas pesquisas são feitas em cima de dados populacionais. Como não existe um modo de prever todos os dados que os usuários necessitarão a cada momento, atualmente utiliza-se a construção de *queries* com linguagem SQL para recuperar informações para pesquisa.

A obtenção de informações gerenciais ou para pesquisa científica exigem conhecimento da base de dados, das relações entre as tabelas, de ferramentas e de

linguagens que não são de domínio dos usuários e pesquisadores em geral, criando uma dependência de pessoal especializado.

Iniciativas estão sendo tomadas no sentido de criar uma base de dados mais apropriada para suporte à obtenção de informação gerencias e de pesquisa. A idéia é a criação de repositório de dados com informações que possam cobrir ambas as necessidades.

O repositório será criado a partir dos dados existentes na base operacional e o processo de criação será feito através da definição de uma estrutura de dados que garanta a ligação entre informações tratadas de forma independente pelos sistemas transacionais. Além disto, a modelagem de dados deste ambiente é desenvolvida objetivando estruturar os dados de forma a permitir a recuperação mais rápida e intuitiva de informações.

1.3. O OBJETIVO

Seja para fazer manutenções ou para obter informações, sem uma metodologia e sem uma ferramenta que ajude a obter informações do código isso se transforma numa tarefa complexa e demorada, levando um tempo que normalmente os usuários não podem esperar. Como estas relações não estão explícitas no esquema, justifica-se a construção de uma ferramenta que possa extrair a regra de integridade referencial do código fonte das aplicações do sistema.

Para extração de informações ou de regras de integridade de um sistema sem documentação pode-se recorrer:

- Ao esquema do banco de dados, para obter: informações das entidades de dados e seus atributos, a integridade referencial quando especificada, a relação entre as tabelas e aplicações e o tipo de acesso que a aplicação faz;
- Aos dados, para obter: seu domínio, cardinalidade entre entidades desde que a relação seja conhecida;
- Ao código fonte das aplicações, para obter: Informações das tabelas e seus atributos, relações entre as tabelas do sistema, relação entre tabelas e aplicações,

regras de negócio, tipo de acesso feito pela aplicação em cada tabela, tabelas internas e seus domínios.

Decidiu-se por usar o código fonte das aplicações para extrair as regras de integridade referencial entre as tabelas do sistema por ser onde podem ser obtidas a maior parte das informações.

O objetivo deste trabalho é mostrar que é possível extrair estas regras a partir do código fonte que as mantém e propor uma ferramenta para extrair, armazenar e recuperar as informações destas relações. Sendo assim, para descrever esse trabalho esta dissertação está organizada em mais seis capítulos, a seguir identificados.

O capítulo 1 destaca a motivação para o desenvolvimento do trabalho, o contexto onde se insere o problema e a contribuição pretendida neste trabalho.

O capítulo 2 descreve as características da regra de integridade que se pretende extrair do código fonte do sistema.

O capítulo 3 descreve os principais aspectos da engenharia reversa na extração de informações do código fonte.

O capítulo 4 apresenta a solução proposta neste trabalho.

O capítulo 5 descreve os resultados obtidos por este trabalho e os trabalhos futuros pretendidos.

Nos anexos estão apresentados os algoritmos utilizados nas diversas fases do trabalho.

2. REGRAS DE INTEGRIDADE

O SIH-HC utiliza o banco de dados IBM/DB2, que é um banco relacional, e por isso é importante entender algumas características deste modelo.

- é simples e fácil de entender com um mínimo de treinamento. Por isso o esquema conceitual pode ser usado por projetistas de base de dados para mostrar o projeto para usuários finais.
- adicionalmente, o esquema conceitual pode ser usado como plano de projeto para que o projetista possa implementar o esquema de dados num SGBD específico.

Para melhor entendimento do modelo relacional, os conceitos básicos estão indicados a seguir [KORTH 1993][CHEN 1976][FAHRNER 1995].

2.1. ENTIDADES

Entidades são o principal objeto deste modelo. São usualmente conceitos reconhecíveis, concretos ou abstratos, tais como pessoas, lugares, coisas, ou eventos e que têm relevância para a base de dados. São classificadas como independentes ou dependentes.

Uma entidade independente é aquela que não tem relação com outra entidade para identificação e uma entidade dependente é aquela que está relacionada a uma outra para identificação. A ocorrência de uma entidade é análoga a uma linha em uma tabela relacional.

Uma entidade pode ser associativa (também conhecida como "intercessão"), usada para associar duas ou mais entidades para resolver uma relação muitos-para-muitos.

2.2. ATRIBUTOS

Atributos descrevem a entidade à qual eles estão associados. Uma instância particular de um atributo é um valor. Por exemplo "David" é um valor do atributo "Nome". O domínio de um atributo é a coleção de valores possíveis que um atributo

pode ter. O domínio de "Nome" é um conjunto de caracteres. Atributos podem ser classificados em identificadores ou descritores. Identificadores são mais comumente chamados de "chaves" (*keys)*, que unicamente identificam uma instância de uma entidade. Um descritor descreve uma característica, não única de uma instância.

2.3. RELAÇÕES

Relações representam uma associação entre duas ou mais entidades. Um exemplo destas relações pode ser:

- Pacientes estão associados a patologias (doenças)
- Unidades atendem especialidades patológicas
- Unidades executam procedimentos

As relações são classificadas em termos de grau, conectividade, cardinalidade e existência.

2.4. CONECTIVIDADE E CARDINALIDADE

A conectividade da relação descreve o mapeamento das instâncias de entidades associadas na relação. Os valores da conectividade são "um" ou "muitos". A cardinalidade de uma relação entre entidades é o número de ocorrências relacionadas para cada uma das duas entidades. Os tipos básicos para uma relação são: um-para-um, um-para-muitos, e muitos-para-muitos.

A relação é um-para-um (1:1) quando pelo menos uma instância da entidade A está associada a uma instância da entidade B.

A relação é um-para-muitos (1:N) quando para uma instância da entidade A, existem zero, uma, ou muitas instâncias da entidade B, mas para a entidade B existe apenas uma instância da entidade A.

A relação é muitos-para-muitos quando para uma instância da entidade A, existem zero, uma, ou muitas instâncias da entidade B e para uma instância da entidade B, existem zero, uma, ou muitas instâncias da entidade A.

2.5. CHAVE PRIMÁRIA E CHAVE ESTRANGEIRA

Chaves primárias e estrangeiras são os componentes básicos nos quais a teoria relacional está baseada. Chave primária força a integridade de entidade pela identificação única de instâncias da entidade. Chaves estrangeiras forçam a integridade referencial porque completam associação entre duas entidades.

2.6. INTEGRIDADE NO MODELO RELACIONAL

2.6.1. Integridade de Dados

Integridade de dados é um dos pontos fundamentais do modelo relacional. Significa que os valores dos dados são corretos e consistentes. A integridade de dados é forçada no modelo relacional pelas entidades e regras de integridade referencial.

2.6.2. Integridade de Entidade

A regra de integridade de entidade estabelece que para toda instância de uma entidade, o valor da chave primária deve existir, ser única, e não nula. Sem a integridade de entidade, a chave primária poderia não preencher esta regra de identificação única para cada instância da entidade.

2.6.3. Integridade Referencial

A regra da integridade referencial estabelece que o valor de toda chave estrangeira tem que ter uma chave primária de igual valor na tabela associada. Integridade referencial garante que podemos navegar corretamente entre entidades relacionadas.

Uma chave estrangeira cria uma relação hierárquica entre duas entidades associadas. A entidade que contém a chave estrangeira é a entidade filho ou dependente, e a tabela que contém a chave primária cujo valor da chave estrangeira aponta é a entidade pai.

Para que a integridade referencial seja mantida entre as entidades pai e filho na inserção ou exclusão de um dado da base de dados, algumas regras de inserção e exclusão devem ser consideradas.

2.6.4. Regras de Inserção

- **Dependente.** Permite inserir uma instância de entidade *filho* apenas se a entidade *pai* já existir.
- **Automática.** Sempre permite inserir a instância de uma entidade *filho*. Se a entidade *pai* não existir, ela é criada.
- **Nula.** Sempre permite inserir a instância de uma entidade *filho*. Se a entidade pai não existir, é atribuído valor nulo à chave estrangeira na entidade filho.
- **Default.** Sempre permite inserir a instância de uma entidade *filho*. Se a entidade *pai* não existir, a chave estrangeira na entidade *filho* assume um valor pré definido.
- **Personalizada**. Permite inserir uma instância de entidade *filho* apenas se uma determinada restrição válida é encontrada.

2.6.5. Regra de Exclusão

- **Restrita.** Permite excluir uma instância de entidade *pai* apenas se não existem instâncias de entidades *filho*.
- **Cascata.** Permite excluir uma instância de entidade *pai* e exclui todas as instâncias de entidade *filho* dependentes.
- **Nula.** Permite excluir uma instância de entidade *pai*. Se existir alguma entidade filho dependente, é atribuído valor nulo às suas chaves estrangeiras.
- **Default.** Permite excluir uma instância de entidade *pai*. Se existir alguma entidade filho dependente, é atribuído um valor pré definido às suas chaves estrangeiras.
- **Personalizada.** Permite excluir uma instância de entidade *pai* apenas se uma determinada restrição válida é encontrada.

2.7. APLICAÇÃO E UTILIZAÇÃO DO MODELO

Um grande número de publicações e ferramentas surgiram desde então para orientar e dar suporte ao projeto e à criação de bases relacionais. Um grande número de bases de dados foi criado em grandes corporações e a maioria em computadores de grande porte. Milhares de linhas de código foram escritas para manter e recuperar

informações destas bases de dados. Pressionados pelo atendimento de solicitações de usuários aliado à necessidade de redução de custos e tempo, profissionais de informática acabaram por comprometer a documentação dos modelos conceituais e lógicos das bases de dados, passando dos levantamentos de dados diretamente para o projeto físico e implementação.

Com o surgimento de novas tecnologias e novos modelos de base de dados, tais como Orientação a Objetos (OO) e microcomputadores, essas grandes corporações passaram a ter novos problemas: alto custo de manutenção e melhorias, tempo excessivamente longo e alto custo para migração dos sistemas para as novas plataformas ou tecnologias. Passaram a ser dependentes dos profissionais que desenvolveram cada parte da sua base de dados ou do longo tempo que um novo profissional levaria para absorver o conhecimento do que não foi documentado [BENEDUSI 1996] [CHIANG 1997] [ANDERSON 1994] [GUEIBER 2000].

Este problema motivou o surgimento de pesquisas, publicações, modelos e ferramentas, para obter informações do modelo conceitual do negócio, fazendo engenharia reversa no sistema existente, como descrito no capítulo 3 a seguir.

3. ENGENHARIA REVERSA

O objetivo da engenharia reversa no campo do *hardware* é a duplicação, enquanto que para o *software* o objetivo é o entendimento suficiente do projeto, para auxílio na manutenção e melhorias. Engenharia reversa por si só não envolve mudanças no sistema ou criação de um novo sistema. É um processo de análise e não um processo de mudança ou replicação [CHIKOFSKY 1990].

Dentro da engenharia reversa podemos destacar duas partes importantes, que são: redocumentação e recuperação do projeto. Redocumentação tem como objetivo criar ou reescrever a documentação do sistema. Já a recuperação do projeto, segundo Ted Biggerstaff, é: "*Recriar abstrações de projeto a partir do código, documentações de projeto existentes, experiências pessoais, conhecimento sobre o problema e domínio da aplicação. Deve reproduzir toda a informação requerida para que uma pessoa possa entender perfeitamente o que um programa faz, como faz e porque faz*".[CHIKOFSKY 1990]

Uma outra definição de engenharia reversa foi dada por R. Arnold: "*O processo de derivação de uma especificação abstrata formal a partir do código fonte de um sistema legado, onde essas especificações podem ser usadas para o processo de engenharia de uma nova implementação deste sistema*" [QUILICI 1995].

Manutenção, de acordo com a ANSI/IEEE Std 729-1993: "*é a modificação de um produto de software, após sua liberação, para corrigir falhas, para melhorias de desempenho ou outros atributos, ou para adaptação do produto a mudanças do ambiente*" e o conceito para engenharia reversa: "*é o processo de análise de um objeto de sistema para: identificar componentes de sistema e suas relações e, criar representações do sistema de uma outra forma ou alto nível de abstração*" [CHIKOFSKY 1990].

Na engenharia reversa o objeto de um sistema pode ser um programa, parte de um programa, um conjunto de programas complexos interagindo, instruções de controle de serviços (*Jobs*), interfaces e arquivos. No processo de engenharia estes

objetos são o resultado do processo de desenvolvimento. Na engenharia reversa estes objetos são o início dos trabalhos.

O sucesso ou fracasso da engenharia reversa de sistemas legados depende dos objetivos específicos do processo da engenharia reversa. Enquanto a maioria dos esforços de engenharia reversa falha ao tentar alcançar os objetivos tradicionais de extrair automaticamente especificações completas para o processo de engenharia, a chance de sucesso seria maior se os objetivos de extração automática de especificações fossem mais modestos, e com suas chances aumentada por assistência humana especializada. [QUILICI 1995]

Isso significa que a chance de sucesso é maior se o foco do problema a ser resolvido for pequeno e bem definido. Também ressalta o fato de que os custos para derivar essas informações do sistema legado, devem ser menores do que o custo para ter essas informações disponíveis para dar suporte ao processo de engenharia.

É preciso que se faça uma avaliação criteriosa de que informações podem ser obtidas automaticamente do código fonte e de que forma elas podem contribuir para o processo de engenharia. As pesquisas atuais enfocam busca de padrões no código que permitam reconhecer abstrações de alto nível, tais como domínio, estrutura de dados e algoritmos, mas infelizmente não é possível extrair um projeto hierárquico completo do mundo real, do sistema legado.

3.1. DESAFIOS PARA ENGENHARIA A REVERSA

Já *Selfridge, Waters e Chikofsky*, [SELFRIDGE 1993], alertam para o fato de que, levado pela importância econômica de manter e implementar grandes bases existentes de sistemas, interesses comerciais e de pesquisa têm crescido rapidamente na última década e que os pesquisadores deveriam estar trabalhando juntos e na direção de metas comuns, tamanha a importância econômica, apresentam dez desafios para aprimoramentos em três áreas: evitar dados artificiais, focar no impacto concreto e facilitar a comunicação entre pesquisadores.

No atual estado da arte existe uma tendência natural de ir na direção de problemas criados para serem solucionados por uma tecnologia particular a ser explorada. Isso limita o progresso obtido com a tecnologia e o impacto econômico da pesquisa porque "problemas criados" têm muito pouco a ver com a realidade, porque "problemas criados" por um pesquisador nada têm a ver com os "problemas criados" por outro pesquisador e os resultados não se transferem.

No artigo são sugeridos alguns desafios para serem seguidos na engenharia reversa e dentre eles alguns foram destacados por estarem mais diretamente em consonância com o propósito deste trabalho. São eles:

- **Evitar a utilização de dados fictícios** - há um risco significativo neste enfoque. Se a simplicidade for longe demais pode acabar em dados que nada têm a ver com a realidade;
- **Usar programas reais como exemplos** – programas criados para validar uma pesquisa costumam ser pequenos (menos de 100 linhas), reduzem as necessidades computacionais, são desenvolvidos pelos próprios pesquisadores ou por estudantes, e não têm nada a ver com programas legados.
- **Usar sistemas como exemplos** - grande número de pesquisas estão focadas em programas. Os problemas reais de engenharia reversa não estão em programas e sim em sistemas. Devemos usar sistemas completos como exemplos mesmo que pequenos, ainda assim deve ser real e coerente com tarefas orientadas para usuários dos sistemas.
- **Tentar soluções úteis acima de qualquer coisa** - Pesquisadores freqüentemente desenvolvem pesquisas, independente de quão promissores elas sejam e de suas aplicações comerciais. Não é fácil para ninguém dizer se o resultado de uma pesquisa de um problema artificial pode ser aplicado a um problema real. Devemos desenvolver ferramentas que tenham aplicação prática, isto é, ferramentas que ataquem problemas reais, de importância econômica e que tragam reais benefícios. Devemos ser dirigidos a problemas e não a tecnologia.

- **Desenvolver aplicações semi-automáticas** - Há um desejo natural de criar ferramentas de engenharia reversa totalmente automáticas. Uma solução completamente automática seria melhor do que uma semi-automática. Entretanto, a engenharia reversa é tão complexa, pelo menos em alguns aspectos, que não parece que muitos dos problemas reais sejam passíveis de uma solução totalmente automática. A interferência humana é altamente desejável e mesmo essencial em termos gerais. Ferramentas de engenharia reversa devem permitir melhorar o processo da engenharia reversa e não serem um fim em si mesmas.
- **Fazer Estudos Empíricos** - Apesar das orientações anteriores, existem pesquisas de engenharia reversa usando dados reais e atacando problemas reais. O que precisamos é de estudos empíricos em programas de sistemas reais em situações reais que mostrem resultados confiáveis.
- **Definir objetivo explícito** - Um fator que complica a comparação entre diferentes enfoques de pesquisadores é que freqüentemente os objetivos não são suficientemente claros. Engenharia reversa é um campo abrangente e pesquisadores têm diferentes opiniões sobre que objetivos devem ser perseguidos. Ao descrever uma ferramenta experimental um pesquisador deve deixar claros alguns pontos. Qual o propósito da ferramenta: entendimento do sistema/programa, redocumentação, reestruturação, recuperação de projeto, plano de testes, extração de parte específica de uma informação global ou alguma outra coisa. Que informações serão recuperadas: especificações, projetos, estrutura de controle, regras de negócio, descrição de interface, arquitetura de dados, algoritmos, etc.

3.2. A EVOLUÇÃO DA ENGENHARIA REVERSA

Pelos idos de 1990, a reengenharia de sistemas legados não era um grande problema, mas recentemente a demanda tem crescido em função do novo direcionamento de sistema para interface *WEB*. A demanda de todos os segmentos de negócio para adaptar seus sistemas de informação para *WEB* criou a necessidade de métodos, ferramentas e infra-estrutura para exploração eficiente destes novos recursos.

A engenharia reversa se transformou em uma das mais promissoras tecnologias no combate do problema de sistemas legados [MULLER 2000].

Nos últimos dez anos, pesquisadores têm produzido facilidades para explorar, manipular, analisar, sumarizar, sintetizar e visualizar componentes de software. Muitas destas ferramentas de reengenharia de software estão focadas na estrutura de sistemas legados com o objetivo de transferir estas informações para os engenheiros de software para ajudá-los na reconstrução ou reuso destas informações.

O investimento em tecnologia para entendimento de programas é crítico para o software e para a indústria da tecnologia da informação a fim de controlar os altos custos e riscos inerentes da evolução de sistemas legados. Engenharia reversa é certamente um excitante campo de pesquisa e pronto para ser ensinado em ciência da computação e em currículos de engenharia de software.

As pesquisas e práticas atuais focam a engenharia e a reengenharia ao nível de código. O processo de engenharia está engajado com qualidade de código. O importante em termos de engenharia reversa é enfatizado nos sistemas legados, onde importantes regras de negócio estão inseridas no código. Durante a evolução do software, são efetuadas mudanças, são acrescentadas novas funções, corrigidos problemas e implementadas melhorias. Em sistemas com documentação pobre, o código é a única fonte de informação confiável do sistema. Como resultado, o processo de engenharia reversa se foca no entendimento do código.

Como sempre, os códigos não contêm todas as informações necessárias. Tipicamente, o conhecimento sobre a arquitetura e regras de negócio, restrições do sistema e domínio das aplicações apenas existem na cabeça dos engenheiros de software, além dos problemas de esquecimentos, saída ou troca de pessoal, depreciação da documentação e aumento da complexidade. Conseqüentemente, isso aumenta a distância entre saber o que é informação útil e as informações de requisitos para alteração do software.

3.3. FERRAMENTAS DE ENGENHARIA REVERSA

Enquanto a necessidade de qualidade e produtividade motivava o surgimento de ferramentas CASE para novos sistemas, os sistemas antigos se transformaram em sistemas legados que continuaram a consumir um volume considerável de tempo e recursos para sua manutenção e melhorias, motivando o surgimento de ferramentas de engenharia reversa.

Este é justamente o caso do SIH-HC, um sistema antigo que qualquer manutenção leva cada vez mais tempo do que se deseja e qualquer pequena melhoria consome cada vez mais tempo e recurso. Na hora de implementar uma modificação no sistema a questão custo benefício esbarra-se no aspecto do tempo necessário para implementá-la. Por isso, uma forma de minimizar o impacto do aspecto tempo, a necessidade da construção de uma ferramenta que auxilie quem vai fazer a modificação a obter informações que diminuam o tempo de busca de informações do sistema.

Embora a engenharia reversa seja um processo de análise de uma aplicação ou sistema para identificar seus componentes e inter-relações, infelizmente algumas pessoas podem entender isso como pirataria, mas em ambientes de grandes corporações, com milhares de programas, com particularidades que interessam apenas à instituição onde foi desenvolvido, e que dificilmente seria adequado à realidade de outras organizações, a engenharia reversa não pode ser confundida com pirataria. Nestas organizações a engenharia reversa é aplicada para melhor entendimento de programas e sistemas, para manutenção e melhorias ou para reengenharia de sistemas [CIFUENTES 2001].

3.4. ESTADO DA ARTE

Pesquisadores têm desenvolvido diversas ferramentas de alto nível para engenharia reversa, ferramentas tais como Rigi [MÜLLER], PBS [FINNIGAN], GXL [HOLT] que é baseada em XML, ou mesmo algumas ferramentas de engenharia reversa de baixo nível, tais como: IDAPro.[GUILFANOV].

A maioria das ferramentas foca apenas um aspecto da engenharia reversa. Elas devem ser especializadas na interpretação de palavras e frases (*parse*) e produzir diferentes tipos de gráficos e diagramas. A principal dificuldade das ferramentas de engenharia reversa começa pela necessidade de dar suporte a uma variedade de linguagens e serem capazes de ser estendidas para outras linguagens. Embora construí-las tenha sido uma tarefa que assusta, ferramentas têm sido projetadas com sucesso para converter COBOL para C, e para traduzir C para C++ ou Java [CIFUENTES 2001].

3.5. COMPARANDO FERRAMENTAS DE ENGENHARIA REVERSA

As funcionalidades das ferramentas variam de capacidade de edição e navegação a relatórios textuais ou gráficos. Existem diversas ferramentas comerciais para engenharia reversa com diferentes funcionalidades e dando suporte a linguagens fonte específicas.

Berndt Bellay e *Harald Gall* [BERNDT 1997], fizeram uma avaliação de quatro ferramentas, segundo critérios pré definidos de funcionalidades, representação, edição/navegação. As ferramentas avaliadas são: Refine/C[1] para engenharia reversa de programas C, usada para entender, avaliar e redocumentar códigos C; Imagix 4D[2] para entender programas C e C++; Rigi é uma ferramenta de domínio público, desenvolvida pelo *Rige Research Projet* na *University of Victoria*, escrita em RCL, uma versão estendida de Tcl/Tk com suporte para análise de fontes C, C++, PL/AS, COBOL e Látex; e Sniff[3] que é uma ferramenta não clássica, aberta, extensível, escalável para ambiente C e C++.

O foco principal da avaliação foi a capacidade de geração de relatórios gráfico do tipo árvores hierárquicas de chamada e gráfico de fluxo de controle e de dados. Um dos resultados da avaliação feita pelos autores pode ser vista na tabela a seguir.

---

1 Refine/C é marca registrada de Reasoning, Inc.
2 Imagix 4D é uma marca registrada de Imagix Corporation
3 Sniff é uma marca registrada de TakeFive Software GmbH

Views	Refine/C	Imagix 4D	Rigi	Sniff+
Estrutura de arquivo		G,T,D		
Construção		G,D		G(limit.)
Chamadas de arquivo		G,D		
Gráfico de chamadas	G	G	G	G
Gráfico de referência cruzada		G		G
Relatório de funções	T	D		
Gráfico de controle de fluxo		G		
Symbol browser				G
Gráfico de fluxo de dados	G	G		
Relatório de variáveis	T	D		
Tipo de relatório / uso, diagrama de estrutura de dados	T	G,D		
Relatório de codificação padrão	T			
Visões em camadas			G	
ShriMP view			G	

Quadro 1 - Tabela comparativa das ferramentas

Estes resultados podem ser relatórios Textuais(T) ou gráficos bi e tri dimensionais (G) e adicionalmente (D) para resultados que podem ser criados como parte de processo automático de documentação.

Extensibilidade da ferramenta é uma característica importante de muitos tipo de ferramentas. Este é o caso para ferramentas de engenharia reversa, em que funcionalidades adicionais freqüentemente precisam ser integradas para resolver restrições específicas da atividade de engenharia reversa.

Refine/C provê uma API para construir ferramentas de análise personalizadas, *Imagix 4D* provê uma linguagem que permite gerar *queries* para usar informações geradas pelo *Imagix 4D* em outras ferramentas, *Rigi* é programável através de uma linguagem de *script* (RCL) e provê uma interface personalizável, *Sniff+* não é extensível.

Como conclusão o artigo coloca que nenhuma das ferramentas pôde ser declarada como melhor. Cada uma delas tem seus pontos fortes e fracos. Todas elas proporcionaram boas capacidades na engenharia reversa em seus diferentes contextos de uso.

A dificuldade enfrentada pelas ferramentas de engenharia reversa é que os sistemas são desenvolvidos em linguagens diversas, mesmo quando se trata de um

mesmo sistema. Outra dificuldade é que o resultado precisa, algumas vezes, ser direcionado para ambientes diferentes. Isto dificulta a criação de ferramentas genéricas e faz com que sejam na sua maioria dirigidas para um escopo limitado.

3.6. AVALIAÇÃO DE FERRAMENTAS DE ENGENHARIA REVERSA

Dentre os aspectos pelos quais uma ferramenta é avaliada estão as perguntas que a ferramenta pode responder quando temos alguma indagação sobre o programa ou sistema, e que para respondermos teríamos que investigar todo o código. Perguntas tais como: que tabelas ou arquivos a aplicação acessa e que tipo de acesso faz? Quais os atributos que a aplicação utiliza de cada tabela ou arquivo? Como é verificada a relação entre as tabelas e como são resguardadas pela aplicação? Qual o tipo ou domínio dos atributos das tabelas acessadas ou de varáveis internas da aplicação?

Muitas outras perguntas poderiam ser relacionadas aqui e todas estariam ligadas à necessidade do entendimento do que a aplicação faz e como faz. Muitas destas indagações podem, e são resolvidas pelos ambientes de desenvolvimento destas aplicações, porém existem algumas que estes ambientes não respondem. São exatamente estas que acabam impactando o processo de entendimento para agilizar as modificações que se deseja implementar, seja como manutenção ou incremento de novas funcionalidades no aplicativo.

Ferramentas de desenvolvimento podem ter um impacto significativo na garantia da qualidade do software e na produtividade do processo de desenvolvimento. Qualidade não é um fator importante apenas no processo de produção mas também durante os estágios do desenvolvimento. Costuma ser uma desculpa que a garantia da qualidade compromete a produtividade. Descobrir como qualidade e produtividade poderiam ser maximizadas no processo de desenvolvimento e manutenção, permanece uma questão aberta no campo da engenharia de software [TORBJORN 1992].

A arquitetura de ferramentas de engenharia reversa deve ter basicamente três categorias de processos: analisar o código fonte, reconhecer a estrutura e semântica do programa, e sublinhar seus componentes, armazenar as informações num repositório e apresentar estas informações para o usuário.

As funcionalidades típicas oferecidas pelas ferramentas de engenharia reversa são: diagrama de fluxo de dados global, diagrama de fluxo de dados modular/inter modular, gráfico de controle, estrutura global de dados, estrutura local de dados, lista de parâmetros de interface entre módulos, lista de parâmetros de cada rotina e uma fração de representação de código fonte correspondente [TORBJORN 1992].

A informação produzida por uma ferramenta de engenharia reversa pode também ser usada para calcular coeficiente e acoplamento ou métricas complexas que têm grande impacto na avaliação da qualidade de software. Outros fatores importantes na qualidade e produtividade de software são os componentes reusáveis. A reusabilidade de componentes de software pode ser medida usando-se ferramentas de engenharia reversa [TORBJORN 1992].

Por se tratar de um problema real a ser resolvido, inicialmente o propósito era implementar a ferramenta VIQUEN, desenvolvida por Ezequiel Gueiber [GUEIBER 2000], que permite a criação de um modelo conceitual gráfico de um esquema relacional com base no esquema de um SGBD, e a partir daí o usuário pode construir pesquisas a uma base de dados com *interface* gráfica de uso intuitivo. No caso da ferramenta o SGBD utilizado foi o ORACLE e o propósito era estendê-la para IBM/DB2 e aplicá-la no ambiente do HC-UFPR.

Como a ferramenta VIQUEN tem com pré-requisito que as regras de integridade estejam definidas no SGBD e como o objetivo era de resolver um problema real do ambiente SIH-HC, o projeto foi direcionado para a recuperação destas informações a partir do código fonte, para que então estas relações de integridade fossem incorporadas às definições da tabelas do SIH-HC no IBM/DB2, tornando-o um banco com inteligência. Isto também permitiria conhecer suficientemente as regras e relações entre as tabelas de um sistema objetivando futuras migrações para outras plataformas ou para criação de banco de dados gerenciais.

3.7. OBTENÇÃO DE REGRAS DE RESTRIÇÃO

O desempenho de um banco de dados, ou mais precisamente a eficiência e a consistência, dependem em grande parte de decisões de projeto. Dependem também,

dentre outras coisas, do conhecimento da semântica do banco de dados, porque as restrições semânticas são pré-requisitos para a normalização e reestruturação de operações, e a determinação formal de restrições semânticas é uma das tarefas mais difíceis no projeto de um banco de dados.[ALBRECHT]

Os problemas na obtenção destas restrições são causados pelas seguintes razões:

- Para a correta utilização de informações de restrição é necessário um profundo conhecimento de lógica. Esta tarefa não apenas demanda altas habilidades de abstração mas também é geralmente muito complexa, principalmente para relações com muitos atributos e em grandes bancos de dados.
- Todos os métodos de projeto precisam de informações completas sobre as regras de restrição para alcançar resultados corretos. Informações incompletas provocam erros ou bancos de dados ineficientes.
- Mesmo que um projetista esteja habilitado a determinar formalmente as regras de restrição, ele pode esquecer algumas por serem muito óbvias ou muito difíceis.
- Algumas restrições são muito complicadas para serem decididas pelo projetista do banco de dados, especialmente se envolvem muitos atributos.

O número de regras de restrição a serem checadas cresce exponencialmente em função do número de atributos.

3.8. CONSIDERAÇÕES

Embora a engenharia reversa de sistemas legados esteja ligada a aspectos custosos para o entendimento do código legado, não podemos abrir mão da busca de soluções que minimizem este impacto, porque engenharia reversa é uma atividade contínua nos ambientes com sistemas legados [WEIDE 1995] [CAGNIN 1999] [AIKEN 1994].

O processo de engenharia reversa precisa ser uma combinação de informações de diversas fontes, independentemente de ser automatizada ou não. Podem ser obtidas a partir dos dados do sistema (dicionário, esquema, instância) [SOUTOU 1998]. Se o

processo for baseado em código fonte passa pelo problema da diversidade de linguagens utilizadas no desenvolvimento de sistemas, dificultando a criação de soluções genéricas o que aponta para soluções individualizadas e dirigidas para a configuração do sistema no contexto onde foi desenvolvido [REUBENSTEIN 1993] [MÜLLER 1993] [MÜLLER 1994] [JARZABEK 1997].

A busca por alternativas conduz a pesquisas de ferramentas já disponíveis e a uma dificuldade natural de avaliação destas soluções. Trabalhos feitos por alguns pesquisadores avaliam algumas destas ferramentas quanto a suas funcionalidades, forma de apresentação de resultados e suas restrições. Dependendo do contexto onde se vai aplicar a solução, seria necessário a combinação de mais de uma ferramenta e dificilmente isso se aplicaria na solução de problemas específicos [GANNOD 1999] [VERMEER 1995] [PREMERLANI 1994] [CHIANG 1995] [JACKSON 1994] [DAVID 1995] [HOLTZBLATT 1997] [RAMESH 1999].

No capítulo 4, a seguir, é apresentada a proposta para as regras de integridade referencial e as etapas seguidas para sua obtenção.

4. PROPOSTA

O objetivo deste trabalho é a extração de regras de integridade referencial entre as tabelas de um sistema a partir do código fonte dos programas que as mantêm. Como na concepção original do SIH-HC foi decidido que não se utilizaria integridade referencial definida no esquema do SGBD do IBM/DB2 [IBM/DB2], esta integridade foi implementada no código dos programas.

Para se conhecer as regras de integridade, ou se conhece o sistema e seu modelo ou esta informação tem que ser buscada no código fonte, o que é uma tarefa complexa e demorada.

O SIH-HC tem pouca ou quase nenhuma documentação e toda vez que um projetista sai da organização ou tem que assumir novas responsabilidades, o novo responsável pelas correções, manutenções e melhorias do sistema precisa obter o máximo de informações sobre o sistema e as regras de negócio para poder continuar atendendo aos usuários do sistema. Sem um modelo ou uma ferramenta que o auxilie nesta tarefa, cada profissional faz isso de maneira subjetiva, procurando por informações onde quer que possa achar. Uma nova mudança de responsável, por qualquer motivo, faz surgir o problema novamente.

Um outro aspecto importante é a necessidade de reengenharia por que vem passando o SIH-HC. Nestes últimos 10 anos, tanto o hardware como software evoluíram muito e o SIH-HC continua usando a mesma plataforma e tecnologia de 10 anos atrás. Isso, na área de informática, significa que o SIH-HC perde em funcionalidade por não se utilizar destas novas tecnologias. Inicialmente cada módulo do sistema foi proposto e desenvolvido por iniciativa do pessoal de informática do HC-UFPR e não por iniciativa dos usuários. A informatização era uma estratégia política definida pela direção do HC. Todas as funcionalidades foram coordenadas pelo assessor de informática e orientadas pelos médicos de cada serviço onde o módulo seria implantado.

Com o passar do tempo, percebeu-se que a racionalização dos processos administrativos não acompanhou a implantação dos sistemas. Na medida que os

usuários foram percebendo que poderiam obter mais benefícios do processo de informatização, a demanda por melhorias e novas funcionalidades cresceu numa proporção maior que a capacidade de atendimento do pessoal da informática.

Para facilitar a continuidade de manutenção e atendimento da demanda dos usuários do sistema, tanto quanto para propiciar facilidades para a reengenharia do SIH, é necessária a implementação de um modelo ou ferramenta que auxilie no processo de engenharia reversa destes módulos. Por isso o objetivo é investigar padrões nos códigos fonte, de modo a permitir a verificação da integridade referencial feita pelos programas.

4.1. ESCOLHA DO SISTEMA

Qualquer um dos dezesseis módulos do SIH-HC poderia ser escolhido para ser utilizado no projeto para extração das regras de integridade referencial. Porém o módulo escolhido foi o de Exames Complementares (EC) exatamente por se tratar de um problema real causado pela saída do projetista original. Desenvolvido para atender Serviços de Apoio Diagnósticos e Terapêuticos (SADT), até a saída do projetista, o módulo estava implantado nos serviços de Radiologia Geral, Tomografia Computadorizada, Endoscopia Digestiva e Medicina Nuclear, faltando ainda ser implantado nos serviços de Anatomia Patológica, Microscopia Eletrônica, Citopatologia, Necropsia, Ultra-sonografia e Endoscopia Per Oral. Este módulo, como os demais módulos do SIH-HC, tem pouca ou quase nenhuma documentação.

4.2. CARACTERÍSTICAS DO SISTEMA

O módulo EC tem 126 programas e 69 tabelas IBM/DB2. Os programas do módulo são divididos em aplicações de manutenção de tabelas básicas e aplicações funcionais. Tabelas básicas são aquelas onde se armazenam códigos e seus significados, tais como tabela de unidades de serviços, tabela de códigos de procedimentos e aplicações funcionais são aquelas que vão efetivamente relacionar as tabelas entre si. Por exemplo: uma requisição de um exame complementar relaciona unidade solicitante, com unidade executante, com paciente e com procedimentos a

serem executados. Desta forma, conclui-se que existe uma relação intrínseca entre as tabelas de unidades, pacientes e procedimentos. Resta saber como os programas garantem estas relações. A figura a seguir ilustra um exemplo destas relações.

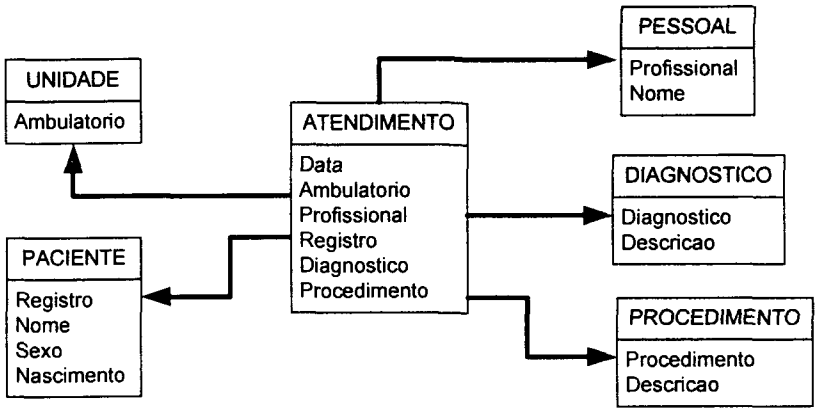

Figura 1 - Exemplo de relação de integridade

A função básica do módulo é permitir que médicos façam requisições de exames complementares via sistema informatizado, quer para pacientes internados ou de atendimento ambulatorial, que os serviços que executam os exames saibam que exames foram requisitados e possam agendar data e hora para coleta e execução do exame. Os laudos são então registrados no módulo, permitindo que o médico possa consultar o resultado em qualquer terminal da rede do HC-UFPR. Cada laudo é parametrizado em função das características de cada serviço ou exame.

4.3. IDENTIFICAÇÃO DOS PADRÕES EXISTENTES

No início do projeto do SIH-HC, foram definidos alguns padrões para o desenvolvimento de aplicações no ambiente IBM/CSP - Cross Structured Program. Estes padrões existem para identificação de tabelas SQL, aplicações, mapas (tela de entrada de dados e relatório), definição de registros, processos, áreas de trabalho internas de aplicação e para identificação de atributos. Estes padrões existem até hoje nos componentes do sistema.

Para cada módulo do SIH-HC foram definidas siglas de identificação e aplicada a cada um de seus componentes como nas tabelas de exemplos, a seguir. Apenas alguns componentes do sistema estão listados a seguir.

DESCRIÇÃO DO SISTEMA	IDENTIFICAÇÃO
Atendimento Ambulatorial	AB
Exames Laboratoriais	LB
Controle de Internação	IT
Controle de Medicamentos de Farmácia	FM
Exames Complementares	EC

Quadro 2 – Sigla de identificação de módulos

TABELA	DESCRIÇÃO
ABAMBULATORIO	Unidades de atendimento ambulatorial
LBREQUISICAO	Requisição de exames clínicos
FMPRESCRICAO	Prescrição de medicamentos
ECAGENDA	Agendamento de exame complementar

Quadro 3 – Padrão para identificação de tabelas SQL

APLICAÇÃO	DESCRIÇÃO
EC00A	Menu Principal
EC02A	Inclusão de Exame Complementar
EC06A	Registro de presença, falta ou cancelamento de agenda
EC20A	Lista Agendas do Dia

Quadro 4 – Padrão para identificação de aplicações

COMPONENTE	DESCRIÇÃO
EC00Wnn	Área de trabalho da aplicação da aplicação EC00A
EC00Pnn	Identificação de processo interno da aplicação
EC00Mnn	Identificação de mapa de tela ou relatório da aplicação
EC00Rnn	Identificação de registro SQL da Aplicação EC00A

Quadro 5 - Padrão para identificação de componentes da aplicação

4.4. PADRÕES DA APLICAÇÃO

O ambiente IBM/CSP que foi utilizado para o desenvolvimento inicial do SIH e o *VisualAge,* que é o ambiente atual, trabalham com componentes onde cada parte da aplicação é construída independente das outras e um módulo principal se encarrega de juntar todos os componentes. Para saber o que uma aplicação faz é preciso ir abrindo cada parte, seguindo a hierarquia de chamadas de cada componente, para analisar o sua lógica.

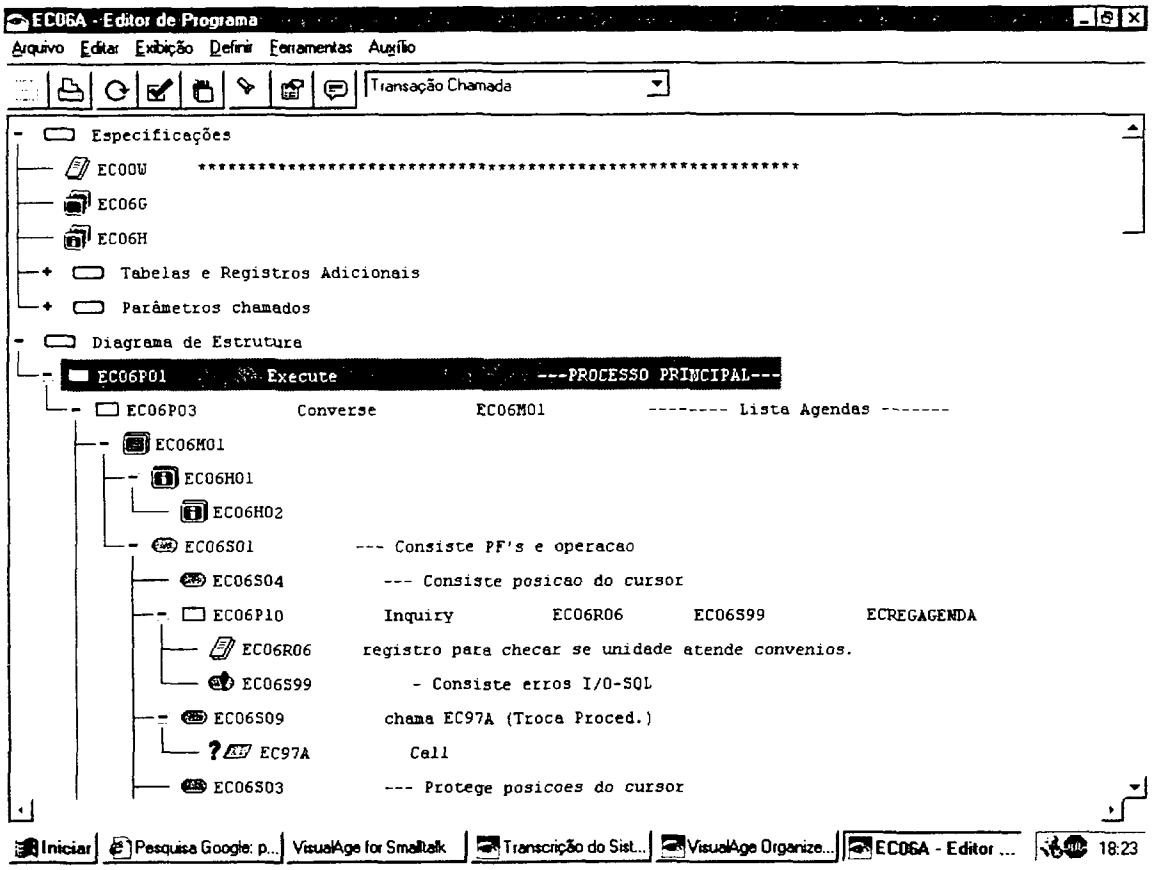

Figura 2 - Apresentação da Aplicação no VisualAge

Para se obter um arquivo com todos os componentes listados em conjunto, se utiliza uma facilidade do próprio ambiente que permite obter um arquivo da aplicação no formato *External Source Format* (ESF) [ESF-REFERENCE], onde todas as partes da aplicação são juntadas e apresentadas no formato texto. Neste formato, cada parte da aplicação pode ser identificada através de palavras-chave (*tags*), próprias do padrão ESF. A seguir estão alguns exemplos destas *tags* e a forma como aparecem no arquivo ESF de uma aplicação.

Nos trechos a seguir identifica-se a aplicação (:appl), e o exemplo de um processo inciado por (:process) e finalizado por (:eprocess).

```
      :appl     name     = EC06A     type     = CALLED

:process   name     = EC06P01                                          00072400
           date     = '06/29/99'          time     = '11:17:59'       00072500
           option   = EXECUTE             refine   = N                00072600
           desc     = '---PROCESSO PRINCIPAL---'.                     00072700
:before.                                                              00072800
;                                                                     00072900
;/* --------- AREA TEMPORARIA PARA TESTES.....                        00073000
;                                                                     00073100
MOVE 1 TO EZECNVCM;                                                   00074600
MOVE 1 TO EZEFEC;                                                     00074700
MOVE 1 TO EZESQISL;                                                   00074800
SET EC06W01 EMPTY;                                                    00074900
;                                                                     00075000
PERFORM EC06P03;                 /* mostra mapa 1                    00075100
;                                                                     00075200
IF EZEAID IS PA2;                /* retorna para menu do sistema     00075300
  MOVE 'S' EC06W07.FLPA2;        /* teclou pa2 -volta menu principal 00075400
END;                                                                  00075500
;                                                                     00075600
EZECLOS;                         /* retorna para apl. chamadora      00075700
:ebefore.                                                             00075800
       :eprocess.
00075900
```

O formato ESF para identificação de registros (**:record**) traz outras importantes informações. A identificação do registro "**name=EC06R06**", a identificação da tabela SQL a que pertence o registro "**:sqltable tableid= eceregagenda**", a identificação de seus atributos "**colname = 'colname'** " e se o atributo é chave da tabela "**key = Y**", mostrado a seguir.

```
:record    name     = EC06R06                                         00227800
           date     = '06/29/99'   time = '11:18:01'                  00227900
           org      = SQLROW                                          00228000
           scope    = GLOBAL                                          00228100
:sqltable  tableid  = 'ecregagenda'                                   00228200
           label    = 'T1'                                            00228300
:recditem  name     = CDTIPO                                          00228700
           scope    = GLOBAL                                          00228800
           colname  = 'CDTIPO'                                        00228900
           readonly = Y          datacode = 453                      00229000
           key      = Y                                               00229100
:recditem  name     = CDIDENT                                         00229200
           scope    = GLOBAL                                          00229300
           colname  = 'CDIDENT'                                       00229400
           readonly = Y                                               00229500
           key      = Y                                               00229600
:recditem  name     = FLREALIZACAO                                    00229700
           scope    = GLOBAL                                          00229800
           colname  = 'FLREALIZACAO'                                  00229900
           readonly = Y          datacode = 453                      00230000
           key      = N                                               00230100
:recditem  name     = FLTROCAPROC                                     00230200
           scope    = GLOBAL                                          00230300
           colname  = 'FLTROCAPROC'                                   00230400
           readonly = Y          datacode = 453                      00230500
           key      = N                                               00230600
:erecord.                                                             00230700
```

Desta forma, é possível selecionar cada parte da aplicação, sobre a qual se deseja obter informações, o que em última análise é feito por quem quer entender o que a aplicação faz.

4.5. VERIFICAÇÃO DAS REGRAS DE INTEGRIDADE

A verificação das regras de integridade entre as tabelas, a partir da análise de aplicações, pode ser feita pela verificação de atributos com o mesmo nome nas tabelas. A partir daí é possível supor que existe relação entre as tabelas com atributos iguais. Acontece que a relação entre tabelas não é feita apenas por atributos com o mesmo nome e atributos com o mesmo nome não garantem que exista uma relação entre tabelas.

Quando uma aplicação faz referência a um atributo de uma tabela, utiliza qualificadores para indicar a tabela à qual o atributo pertence. Desta forma, uma outra maneira de verificar regras de integridade entre as tabelas é pela análise dos comandos IF e MOVE quando relacionam atributos de tabelas diferentes. Não faz sentido uma aplicação comparar ou mover atributos de tabelas diferentes, sem que estes atributos sejam do mesmo domínio. Desta forma, se uma aplicação compara dois atributos de tabelas diferentes ou move um atributo de uma tabela para um atributo de outra tabela, podemos garantir que existe uma relação entre estas tabelas. Um exemplo de como isso pode ser feito está na análise do trecho de aplicação a seguir.

```
IF EC06R01.CDUNIDEXEC NE EC06R19.CDUNIDADE;                    00246500
  MOVE EC06R01.CDUNIDEXEC TO EC06R19.CDUNIDADE;                00246600
  MOVE 'E' TO EC06R19.TPUNIDADE; /* Unidade executante        00246700
  PERFORM EC06P31;           /* inquiry ECUSUUNID              00246800
END;                                                           00246900
IF EC06R01.CDPROCEDIMENTO NE EC06R21.CDPROCEDIMENTO;           00247000
  MOVE EC06R01.CDPROCEDIMENTO TO EC06R21.CDPROCEDIMENTO;       00247100
  PERFORM EC06P06;           /* inquiry ECPROCED               00247200
  IF EC06R21 IS NRF;         /* NRF                            00247300
    MOVE 39 TO EC06W07.NRMSG; /* problema de integridade       00247400
    EZECLOS;                                                   00247500
  END;                                                         00247600
END;                                                           00247700
```

No trecho, a aplicação compara os atributos EC06R01.CDUNIDEXEC e EC06R19.CDUNIDADE. Como o padrão de registros EC06R01 e EC06R19 é usado para referenciar tabelas e as tabelas podem ser identificadas, a partir do trecho

podemos deduzir que existe uma relação entre esses dois registros através dos atributos CDUNIDEXEC e CDUNIDADE. A análise da aplicação mostra que isso é verdade porque ambos tratam de unidades ou serviços da estrutura do HC-UFPR e onde um serviço solicita um exame para ser executado por um outro serviço.

No mesmo trecho aparece uma relação entre EC06R01 e EC06R21, onde este último refere-se a uma tabela de procedimentos possíveis de serem realizados. A mesma consideração podemos fazer em relação ao atributo CDPROCEDIMENTO, comum nos dois registros.

Para buscar outras relações possíveis, a mesma coisa podemos fazer considerando o comando MOVE.

4.6. EXTRAÇÃO DE INFORMAÇÕES

O primeiro passo é extrair de cada aplicação quais tabelas ela acessa, quais atributos utiliza e quais são chave da tabela. Como os padrões dessas informações já estão identificados, pode-se usar comandos de manipulação de texto para identificar e extrair essas informações. Para testar isso, foram usadas ferramentas do ambiente Unix/Linux, tais como *grep*, *sed* e *awk*. Um exemplo para identificação e extração de padrões está a seguir.

```
egrep -H ':record|:sqltable|:recditem|colname|key      =|:erec' EC06A
```

A saída do comando acima nos dá uma lista como o exemplo a seguir, com todos os registros da aplicação (na lista apenas um registro está indicado):

```
EC06A::record    name      = EC06R06                    00227800
EC06A::sqltable  tableid   = 'ecregagenda'              00228200
EC06A::recditem  name      = CDTIPO                     00228700
EC06A:           colname   = 'CDTIPO'                   00228900
EC06A:           key       = Y                          00229100
EC06A::recditem  name      = CDIDENT                    00229200
EC06A:           key       = Y                          00229600
EC06A::recditem  name      = FLREALIZACAO               00229700
EC06A:           colname   = 'FLREALIZACAO'             00229900
EC06A:           key       = N                          00230100
EC06A::recditem  name      = FLTROCAPROC                00230200
EC06A:           colname   = 'FLTROCAPROC'              00230400
EC06A:           key       = N                          00230600
EC06A::erecord.                                         00230700
```

Esta lista pode ser tratada para colocar todas as informações de um registro numa mesma linha e se isso for feito para todas as aplicações, obtêm-se uma lista de todos os registros das aplicações, a quais tabelas se referem, quais os seus atributos e destes, quais são usados como chave de acesso.

O segundo passo é identificar as relações entre tabelas. Como os comando IF e MOVE garantem a existências de uma relação entre tabelas, é a próxima informação a ser obtida. Da mesma forma como foi identificado e selecionado o registro, pode-se extrair esse comando da aplicação com a utilização das mesmas ferramentas de manipulação de texto do ambiente Unix/Linux. Com isso as relações serão conhecidas pelos pares de comandos identificados.

O terceiro passo é guardar estas informações para que sejam recuperadas e também confirmadas posteriormente.

4.7. VERIFICAÇÃO DE INTEGRIDADE REFERENCIAL

O tratamento isolado da integridade referencial parece uma coisa difícil de ser implementada, porque cada programador tem seu estilo de programação e alguns padrões podem não ser identificados no código. Entretanto a combinação de outras informações mais fáceis de serem obtidas, pode ajudar a contemplar esta deficiência.

Uma das formas pelas quais as aplicações verificam integridade referencial já foi mostrada no mesmo trecho de programa onde se verificam as relações entre tabelas pelos comandos IF e MOVE. A aplicação usa a sintaxe "IS NRF" (Not Record Found), para testar se uma referência de um atributo existe em outra tabela. Quando esta cláusula vem seguida do comando (EZECLOS), que encerra a aplicação, podemos concluir então que uma relação obrigatória entre duas entidades foi quebrada e com isso a integridade referencial do banco foi comprometida.

Como a aplicação EC06A, do qual o trecho citado faz parte, tem função conhecida, que é a de registro de presenças ou faltas de pacientes para uma agenda de coleta de material para exame, não faz sentido existir uma agenda que se refere a um

procedimento não cadastrado na tabela de procedimentos e por isso a aplicação se encerra se encontrar esta situação.

Este padrão é um pouco mais difícil de identificar nas aplicações porque a identificação das tabelas envolvidas, bem como o comando de encerramento, podem não estar todos de uma mesma forma em todas as aplicações. Depende do estilo de programação de cada programador. Como os padrões estão lá, também podem ser identificados e extraídos da aplicação.

Regras de integridade referencial são verificadas pela relação de atributos chaves entre as tabelas. Como foi visto nos itens 4.5 e 4.6, é possível identificar quais registros definem quais tabelas, quais atributos são chave e como identificar as relações entre atributos através dos comandos MOVE e IF. Se numa destas relações, um dos atributos é chave, então a relação é uma relação de integridade referencial. Desta forma é possível extrair as regras de integridade referencial a partir do código fonte, dentro das condições e requisitos apresentados.

4.8. EXTRAINDO INFORMAÇÕES

Os padrões, regras de extração e forma de armazenamento foram implementados em três algoritmos em *Perl*. O primeiro, para extrair informações das tabelas acessadas pela aplicação (Anexo A). O segundo, para extrair informações de relações entre tabelas baseadas nos comandos MOVE das aplicações (Anexo B). O terceiro para extrair informações de relações entre tabelas baseadas nos comandos IF das aplicações (Anexo C). A escolha da linguagem baseou-se apenas no fato de ser ela de conhecimento do programador e por possuir recursos para identificar, extrair e armazenar os padrões necessários das aplicações. É sabido que poderiam ser utilizadas outras linguagens, algumas com mais ou menos recursos para manipulação de textos e permitiriam obter os mesmos resultados.

4.9. ARMAZENAMENTO DAS INFORMAÇÕES

Para armazenar as informações obtidas dos códigos fonte, decidiu-se pelas tabelas SQL, que permitem a construção de *queries* SQL para obtenção de

informações. O SGBD utilizado foi o MySQL, por ser de utilização livre. Para isso foram definidas as seguintes tabelas: a) tabTables – para armazenar as informações das tabelas; b) tabFields – para armazenar informações dos atributos e suas ligações com tabelas; c) tabViews – para armazenar as visões (registros) de cada tabela pelas aplicações do sistema; d) tabVwFields – para armazenar os atributos participantes de uma View e e) TabVwRelations – para armazenar as relações identificadas nas aplicações e os atributos envolvidos.

Tab. MySql para armazenar dados de tabelas

```
##
# tabTables - Informações de Tabelas
#
CREATE TABLE tabTables (
    tabId integer unsigned auto_increment primary key,
    tabSchema varchar(128),
    tabName varchar(128),
    tabRemarks varchar(254),
    tabCount integer unsigned default 1
);
```

Tab. MySql para armazenar dados de atributos

```
##
# tabFields - Atributos e suas ligações com as tabelas
#
CREATE TABLE tabFields (
    fieId integer unsigned auto_increment primary key,
    tabId integer unsigned,                # tabTables->tabFields
    fieName varchar(128),
    fieType varchar(18),
    fieLength integer,
    fieKey smallint,
    fieRemarks varchar(254),
    fieCount integer unsigned default 1
);
```

Tab. MySql para armazenar Views das aplicações

```
##
# tabViews - Identificação das Views e aplicações referenciadas
#
CREATE TABLE tabViews (
    vieId integer unsigned auto_increment primary key,
    tabId integer unsigned,                    # tabTables->tabViews
    vieName varchar(128),
    vieApp varchar(128),
    vieRemarks varchar(254)                    # Prolog from 'esf'
);
```

Tab. MySql para armazenar os campos de uma View

```
##
# tabVwFields - Identificação dos atributos que participam de uma View
#
CREATE TABLE tabVwFields (
    vfiId integer unsigned auto_increment primary key,
    vieId integer unsigned,                    # tabViews->tabVwFields
    fieId integer unsigned,                    # tabFields->TabVwFields
    vfiName varchar(128),                      # View column name
    vfiKey enum('Y', 'N') default 'N',         # Is key
    vfiRemarks varchar(254)                    # Remarks from 'esf'
);
```

Tab. MySQL para armazenar relações

```
##
# tabVwRelations - Identifica as relações entre atributos e suas Views
#
CREATE TABLE tabVwRelations (
    vreId integer unsigned auto_increment primary key,
    relId integer unsigned,                          # tabRel->tabVwRel
    vfiId0 integer unsigned,                         # vfiId0 <-> vfiId1
    vfiId1 integer unsigned,                         # vfiId1 <-> vfiId0
    vreName varchar(128),                            # View relation name
    vreType enum ('move', 'if'),                     # Relation get from
    vreMode enum ('normal', 'fk') default 'normal'   # 'fk' = foreign key
);
```

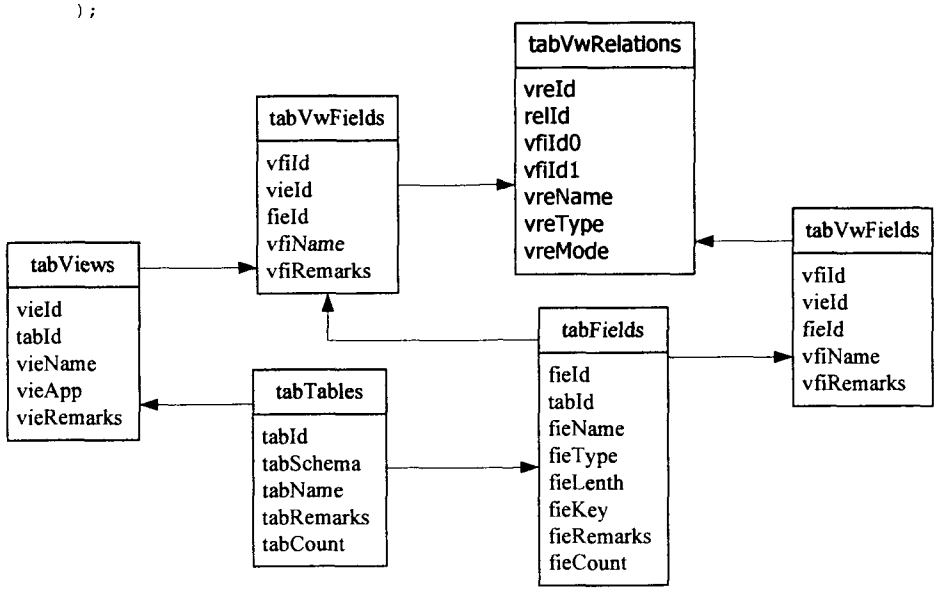

Figura 3 - Modelo Conceitual para Armazenamento

5 RESULTADO

Uma vez armazenadas as informações extraídas das aplicações, elas podem ser agora obtidas a partir de comandos SQL ou de *queries* previamente preparadas para obtenção das informações. A partir do esquema do SGBD pode-se obter a lista das tabelas e seus atributos de um determinado sistema. Como no SIH-HC as tabelas seguem o padrão estabelecido para nomes de tabela, apenas as tabelas com o mesmo prefixo seriam obtidas. Neste caso, se desejássemos obter a relação das tabelas do sistema de Exames Complementares, seriam obtidas apenas as tabelas que começassem com "EC".

Analisando o resultado da extração das informações de tabelas das aplicações, verificou-se que o subsistema "EC" utiliza tabelas de outros sistemas. Uma informação que o esquema não dá é a relação entre as tabelas de um sistema, a menos que elas estejam definidas nas relações de integridade do SGBD. Mesmo assim, as relações entre tabelas que não envolvam integridade, não são identificadas no esquema. Outra informação interessante que pôde ser obtida é o número de referências feitas pelas aplicações a cada tabela ou atributo, como pode ser visto no (Anexo D).

Considerando que cada registro de uma aplicação contém apenas os atributos que a aplicação utiliza de cada tabela, estes registros podem então ser considerados "visões" que a aplicação tem da tabela. Desta forma podemos identificar todas as "visões" que referenciam uma mesma tabela. Uma lista das visões de uma determinada tabela pode ser vista a seguir, identificando para uma mesma tabela (tabId=90), quais as aplicações que a acessam (vieApp) e a identificação de cada "visão" (vieView).

```
SELECT t1.tabName, t1.tabId, t2.vieApp, t2.vieView
FROM tabTables t1, tabViews t2
WHERE t1.tabId = t2.tabId AND t1.tabName = 'ECAGENDA';
+----------+-------+---------+---------+
| tabName  | tabId | vieApp  | vieView |
+----------+-------+---------+---------+
| ECAGENDA |    90 | EC02A   | EC02R20 |
| ECAGENDA |    90 | EC04A   | EC04R02 |
| ECAGENDA |    90 | EC05A   | EC05R07 |
| ECAGENDA |    90 | EC06A   | EC06R01 |
| ECAGENDA |    90 | EC17A   | EC17R07 |
| ECAGENDA |    90 | EC19A   | EC19R01 |
| ECAGENDA |    90 | EC34A   | EC34R02 |
| ECAGENDA |    90 | EC34A   | EC34R05 |
| ECAGENDA |    90 | EC54A   | EC54R01 |
| ECAGENDA |    90 | EC70A   | EC70R08 |
| ECAGENDA |    90 | EC72A   | EC72R09 |
| ECAGENDA |    90 | EC74A   | EC74R01 |
| ECAGENDA |    90 | EC89A   | EC89R07 |
+----------+-------+---------+---------+
```

Uma outra informação que pode ser obtida do esquema, mas agora também das informações armazenadas, é a referência cruzada entre aplicações e tabelas. A relação a seguir mostra todas as tabelas acessadas pela aplicação EC06A e quantas vezes cada tabela é referenciada dentro da aplicação.

```
SELECT t2.vieApp, t1.tabName, t1.tabId, t1.tabCount
FROM tabTables t1, tabViews t2
WHERE t1.tabId = t2.tabId AND t2.vieApp = 'EC06A'
+--------+----------------+-------+----------+
| vieApp | tabName        | tabId | tabCount |
+--------+----------------+-------+----------+
| EC06A  | ECREGAGENDA    |   113 |       16 |
| EC06A  | ECAGENDA       |    90 |       17 |
| EC06A  | ECUSUUNID      |   115 |       22 |
| EC06A  | ECPROCED       |   111 |       20 |
| EC06A  | ECREALPROC     |   467 |       18 |
| EC06A  | ECREALPROC2    |   653 |       13 |
| EC06A  | HCPROCEDIMENTO |   204 |       49 |
| EC06A  | HCLOTACAO      |   520 |       48 |
| EC06A  | ECMOVAGENDA    |   109 |       21 |
| EC06A  | ECREQUISICAO   |   114 |       23 |
| EC06A  | ECCAPAGENDA    |    91 |        7 |
+--------+----------------+-------+----------+
```

Uma outra informação que pode ser obtida é a relação de tabelas acessadas pelo sistema EC. Como foi dito antes, o esquema não consegue identificar todas as tabelas referenciadas por um sistema porque não conhece como as aplicações relacionam estas tabelas. Como as tabelas acessadas pelo sistema podem ser recuperadas das aplicações, a relação pode mostrar todas as tabelas referenciadas, mesmo que pertençam a outros módulos.

Relação de todas as tabelas acessadas pelo módulo "EC" e quantas vezes cada uma foi referenciada pelas aplicações: (aqui listada apenas uma parte)

```
SELECT tabName, tabCount FROM tabTables
+------------------+----------+
| tabName          | tabCount |
+------------------+----------+
| ABAGENDA         |        2 |
| ABAMBULATORIO    |       19 |
| ABCALENDARIO     |        9 |
| ABSAM            |        2 |
| AGMOV_EXAME      |        2 |
| ECAGENDA         |       18 |
| ECCAPAGENDA      |        8 |
| ECCAPATEND       |        9 |
| ECCOLREQ         |       12 |
| ECCOMPLAUDO      |       12 |
| ECCOMPREQ        |        8 |
| ECCONCLUSAO      |       10 |
| ECCONCLUSAO1     |        9 |
| ECCONCLUSAO2     |       10 |
| ECCONTCOLREQ     |        6 |
| ECCUIDADO        |       12 |
| ECFILME          |        6 |
+------------------+----------+
```

A informação que mais se buscava era exatamente as relações entre as tabelas do módulo. Partindo das informações obtidas nos códigos fonte, é possível obter uma lista das relações entre tabelas de um módulo, os atributos envolvidos, de onde foi obtida a relação e se os atributos envolvidos são ou não chave, como no exemplo a seguir: (apenas algumas relações foram listadas aqui)

```
SELECT t4.tabId, t5.tabID, t1.vfiName, t2.vfiName, t3.vreType, t3.vreMode
FROM tabVwFields t1, tabVwFields t2, tabVwRelations t3, tabViews t4, tabViews t5
WHERE t1.vfiId = t3.vfiId0 AND t2.vfiId = t3.vfiId1 AND t1.vieId = t4.vieId
AND t2.vieId = t5.vieId
order by 1,2;
+-------+-------+----------------+----------------+---------+---------+
| tabId | tabID | vfiName        | vfiName        | vreType | vreMode |
+-------+-------+----------------+----------------+---------+---------+
|    90 |    16 | NRREGISTRO     | REGISTRO       | move    | normal  |
|    90 |   111 | CDPROCEDIMENTO | CDPROCEDIMENTO | move    | normal  |
|    90 |   111 | CDPROCEDIMENTO | CDPROCEDIMENTO | if      | fk      |
|    90 |   113 | CDUNIDEXEC     | CDIDENT        | move    | normal  |
|    90 |   114 | NRREGISTRO     | NRREGISTRO     | move    | normal  |
|    90 |   114 | DTHRREQUISICAO | DTHRREQUISICAO | move    | normal  |
|    90 |   115 | CDUNIDEXEC     | CDUNIDADE      | move    | normal  |
|    90 |   115 | CDUNIDEXEC     | CDUNIDADE      | if      | normal  |
|    90 |   204 | CDPROCEDIMENTO | CDPROCEDIMENTO | if      | fk      |
|    90 |   204 | CDPROCEDIMENTO | CDPROCEDIMENTO | move    | normal  |
|    90 |   213 | NRREGISTRO     | REGISTRO       | if      | fk      |
|    90 |   520 | CDUNIDEXEC     | CDLOTACAO      | move    | normal  |
|    90 |   520 | CDUNIDEXEC     | CDLOTACAO      | if      | fk      |
|    91 |   537 | CAPNORMALAB    | CAPNORMALAB    | if      | normal  |
|    91 |   537 | CAPCONVENIOAB  | CAPCONVENIOAB  | if      | normal  |
|    91 |   537 | CAPNORMALIT    | CAPNORMALIT    | if      | normal  |
|    91 |   537 | CAPCONVENIOIT  | CAPCONVENIOIT  | if      | normal  |
|    93 |    99 | CDTIPO         | CDTIPO         | move    | normal  |
|    93 |    99 | CDIDENT        | CDIDENT        | move    | normal  |
|    93 |    99 | CDTIPO         | CDTIPO         | move    | normal  |
|    93 |    99 | CDIDENT        | CDIDENT        | move    | normal  |
|    94 |    96 | CDCONCLUSAO    | CDCONCLUSAO    | move    | normal  |
|    94 |    97 | CDCONCLUSAO1   | CDCONCLUSAO1   | move    | normal  |
|    94 |    98 | CDCONCLUSAO2   | CDCONCLUSAO2   | move    | normal  |
|    94 |   108 | DTHRREQUISICAO | DTHRREQUISICAO | move    | normal  |
|   107 |   102 | CDUNIDEXEC     | CDUNIDEXEC     | move    | normal  |
|   107 |   102 | INDCLINICA     | DSDIAGNOSTICO  | move    | normal  |
|   107 |   102 | CDUNIDSOLIC    | CDAMBSOLIC     | move    | normal  |
|   107 |   115 | CDUNIDEXEC     | CDUNIDADE      | move    | normal  |
+-------+-------+----------------+----------------+---------+---------+
```

6 CONCLUSÃO

6.1 QUANTO AOS OBJETIVOS

O objetivo do trabalho foi atingido e de fato é possível extrair e armazenar regras de integridade referencial das tabelas de um esquema a partir do código fonte das aplicações. Os padrões inicialmente definidos para o desenvolvimento dos sistemas bem como os padrões dos códigos fonte no formato ESF, facilitaram a extração das regras de integridade e a solução mostrou-se não ser dependente destes padrões.

6.2 QUANTO AOS RESULTADOS

Os resultado obtidos foram armazenados em tabelas MySql, mas poderiam ser utilizados outros SGBDs ou planilhas, e mesmo arquivos texto. A alternativa escolhida permite a construção de *queries* ou a utilização de *queries* previamente preparadas para recuperar informações.

Durante o desenvolvimento da ferramenta foram obtidas também informações do esquema para conferir com as informações obtidas pela ferramenta. Como resultado, constatou-se que as informações sobre as tabelas, seus atributos e identificação de atributos chave estavam corretas.

O esquema (IBM/DB2) fornece informações sobre relação entre tabelas e aplicações, o que também foi obtido pela ferramenta. O esquema fornece informações sobre o tipo de acesso (select, insert, delete, update) que cada aplicação faz a cada tabela e, embora isso não tenha sido implementado na ferramenta, seria possível obter porque existem padrões no código que permitem que essa informação seja obtida.

O esquema (IBM/DB2) fornece informações das relações entre tabelas desde que elas sejam identificadas por integridade referencial no esquema. Como a integridade referencial está presente no código das aplicações, elas puderam ser recuperadas do código fonte das aplicações e não do esquema.

O esquema (IBM/DB2) fornece informações sobre visões para o banco de dados, desde que sejam registradas no esquema, mas como isso também não existe no ambiente do SIH-HC, essas informações puderam ser obtidas pela ferramenta.

O esquema (IBM/DB2) fornece informações sobre a relação entre as aplicações e os atributos de cada tabela através dos módulos de acesso, mas o número de vezes que um atributo ou uma tabela é referenciado numa aplicação ou no sistema EC foi obtido pela ferramenta.

6.3 TRABALHOS FUTUROS

Entre futuras implementações para esta ferramenta estão:

- extração de informações de mapas de telas e relatórios;
- obtenção de mapa hierárquico de chamadas de aplicações;
- obtenção de cardinalidade das tabelas pela análise dos dados do banco;
- obtenção de dados de tabelas internas das aplicações, utilizadas para validação de dados;
- Obtenção do fluxo de dados entre aplicações.
- Disponibilização de informações via browser web.

REFERÊNCIAS

AIKEN, P.; MUNTZ, A.; RICHARDS, R. **DoD Legacy Systems Reverse Engineering Data Requirements**. COMUNICATIONS OF THE ACM. May 1994 / Vol. 37

ALBRECHT, A.; BUCHHOLZ, E.; DÜSTERHÖFT, A.; THALHEIN, B. **A Tool for Obtaining Semantic Constraints,**

ANDERSON, M. **Extracting an Entity Relationship Schema from a Relational Database through Reverse Engineering**, Entity-Relationship Approach - ER'94: Bussiness Modelling and Re-Engineering, P. Loucopoulos, (ed.), (LCNS 881) (Springer-Verlag, 1994) 403-419.

BENEDUSI, P. **Improving reverse engineering models with test-case related knowledge**, Information and Software technology 38 (1996) 711-718

BERNDT, B.; GALL, H. **A Comparision of four Reverse Engineering Tools**, IEEE 1997

CAGNIN, M.; PENTEADO, R. Avaliação das vantagens quanto à facilidade de manutenção e expansão de sistemas legaods sujeitos à engenharia reversa e segmentação. Tese de Mestrado. UNIVERSIDADE FEDERAL DE SÃO CARLOS. Departamento de Computação. São Paulo, Brasil: [s.n.], 1999

CHEN, P. **The Entity-Relationship Model** – Toward a Unified View of Data, ACM Transaction on Database Systems, Vol. 1, March 1976, Pages 0-36.

CHIANG, R.; BARRON, T. **Quality Issues in Database Reverse Engineering: An Overview.** Proceedings of the 1995 IEEE Annual International Engineering Management Conference, pp. 185-189, June, 1995

CHIANG, R.; BARRON, T., STOREY, V., **A framework for the design and evaluation of reverse engineering methods for relational databases**, Data& Knowledge Engineering 21 (1997) 57-77.

CHIKOFSKY, E.; CROSS II,J. **Reverse Engineering and Design Recovery: A Taxonomy** IEEE Software pp. 13-17 Jan. 1990.

CIFUENTES, C. **Reverse Engineering and Computing Profession**, IEEE The Profession, December 2001

CSP - **Cross System product – General Information** – IBM GH23-0500-4, Fifth Edition (May 1990)

DAVIS. K. **August-II: A Tool for Step-byStep Dta Model Reverse engineering.** 2nd Working Conference on Reverse Engineering, WCRE '95, July 14-16, 1995, Toronto, Canada. IEEE Computer Society Press, 1995, ISBN 0-8186-7111-4.

ESF- REFERENCE - **IBM VisualAge Generator, External Source Format Reference** – SH23-6609-01, Versão 3.1, Second Edition (1998)

FAHRNER, C.; VOSSEN, G. **A survey on database design transformations based on the Entity-Relationship model**, Data Knowledge Eng. 15 (1995) 213-250.

FINNIGAN, P. **PBS Portable Bookshelf.** UNIVERSITY OF WATERLOO. Disponivel em:<http://swag.uwaterloo.ca/pbs> Acesso em:.22 ago. 2002.

GANNOD, G.; CHENG, B. **A Framework for Classifying and Comparing Software Reverse Engineering and Desogn Recovery Thecniques.** In Proceedings of the 6th Working Conference on Reverse Engineering (WCRE). IEEE CS Press, October 1999

GUEIBER, E. **VIQUEN - Um Ambiente Interativo para Consulta Visual e Extração de Esquemas**, Tese de Mestrado. UNIVERSIDADE FEDERAL DO PARANÁ. Departamento de Informática. Paraná Brasil: [s.n.], 2001.

GUILFANOV **IDA Pro Disassembler.** Disponivel em:<http://www.datarescue.com/idabase/ida.htm> Acesso em 22 ago. 2002.

HOLT, R. **GXL Graph eXchange Language**. UNIVERSITY OF WATERLOO. Disponivel em:<http://www.gupro.de/GXL> Acesso em 22 ago.2002.

HOLTZBLATT, L.; PIAZZA, R.; REUBENSTEIN, H.; ROBERT, S.; HARRIS, D. **Design Recovery for Distributed Systems.** . IEEE Transactions on Software Engineering, 23(7):461--472, July 1997..

IBM/DB2 - **DB2/VM Database Administration**, IBM GC092388-00

JACKSON, D.; ROLLINS, E. **A New Model of Program Dependences for Reverse Engineering. Proceedings of the second ACM SIGSOFT symposium on Foundations of software engineering. 1994 , New Orleans, Louisiana, United States**

JARZABEK, S.; WOON, I. **Towards a Precise Description of Reverse Engineering Methods and Tools**. 1st Euromicro Working Conference on Software Maintenance and Reengineering (CSMR '97) March 1997 Berlin, GERMANY. 1997 IEEE 0-8116-7892-5/97

KORTH, F.; SILBERSCHATZ, A.; **Sistema de Banco de Dados**, 2ª ed. – São Paulo: MAKRON Books, 1993

MÜLLER, H. **A Visual Tool for understanding legay systems.** UNIVERSITY OF VICTORIA, BC, Canada. Disponivel em:<http://www.rigi.csc.uvic.ca> Acesso em: 22 ago. 2002.

MÜLLER, H.; JAHNKE, J.; SMITH. D.; STOREY, A.; TILLEY, S.; WONG, K. **Reverse Engineering: A Roadmap.** ACM New York, USA – 2000, 47-60 - ISBN:1-58113-253-0

MÜLLER, H.; TILLEY, S.; UHL, J. **A Reverse Engineering Approach To Subsystem Structure Identification.**. Journal of Software Maintenance: Research and Practice, 5(4), pages 181-204, December 1993.

MÜLLER, H.; WONG, K.;TILLEY, S. **Understanding Software Systems Using Reverse Engineering Technology.** In Proc. of the 62nd Congress of L'Association Canadienne Francaise pour l'Avancement des Sciences, (ACFAS) , pages 41--48, Montreal, PQ, 16--17 May 1994

PREMERLANI, W.; BLAHA, M. **An Approach for Reverse Engineering of Relational Databases.** COMUNICATIONS OF THE ACM. May 1994 / Vol. 37

QUILICI, A. **Reverse Engineering of Legacy Systems: A Path Toward Success** - Proceedins of the 17th Annual International Conference on Software Engineering, IEEE Press. Seattle, WA, pp. 331-336, Abril 1995

RAMESH, V.; BROWNE, G. **Expressing casual relationship in conceptual database schemas.** Journal of Systems and Software, 45, 1999, pp. 225-232. Elsevier Science 1999.

REUBENSTEIN, H.; PIAZZA, R.; ROEBERTS, S. **Separating Parsing and Analysis in Reverse Engineering Tools.** Proceedings of Working Conference on Reverse Engineering, WCRE 1993, May 21-23, 1993, Baltimore, Maryland, USA. IEEE Computer Society Press

SELFRIDGE, P.; WATERS, C.; CHIKOFSKY, E. **Challenges to the Field of Reverse Engineering.** IEEE 0-8186-3780-3/93, 1993

SOUTOU, C. **Relational database reverse engineering: Algorithms to estract cardinality constraints**. Data & Knowledge engineering 28– Elsevier Science. 1998 161-207

SQL/DS - **General Information for SQL/DS 3.2** (IBM GH09-8074-01); Version date: SEP 90; **SQL/DS 3.4 General Information for VM** (IBM GH09-8074-02) Version date: FEB 93

TORBJORN, S.; KHAN. K. **Assessment of ReverseEngineering Tolls**: A MECCA Approach, IEEE 1992

VAGen - **VisualAge Generator**, ISBN: 0738400831, IBM Form Number: SG24-4238-00, 31-December-1996.

VERMEER, M.; APERS, P. **Reverse engineering of relational database applications.** Proceedings Fourteenth International Conference on Object-Oriented and EntityRelationship Modeling (ER'95), Gold Coast, Australia, M. P. Papazoglou, ed., SpringerVerlag, New York--Heidelberg--Berlin, December 1995, 89--100

WEIDE, B.; HEYM, W. **Reverse Engineeringof Legacy Code Exposed.** Proceedings 17th International Conference on Software Engineering, ACM, April 1995, pp. 327-331

ANEXOS

Anexo A – Algoritmo para extração das tabelas e atributos a partir do código fonte das aplicações

```
##
# Main session
#-------------------------------------------------------------------------------

$leaf = \%tree;                          # get pointer to %tree
while (<>) {                             # hein funny, 'long life to perl'
    $line++;                             # increment line counter
    if (/^:appl\s*name\s*=\s(\w*)/) {    # test ':appl'
   $appl = $1;                    # $1 = application name
   /type\s*=\s(\w*)/;             # get type of application
   $appltype = $1;                       # $1 = application type
    }
    elsif (/^:record\s.*=\s(\w*)/) {     # test ':record'
        $leaf->{'line'} = $line;
   $leaf->{'appl'} = $appl;       # get current application
   $leaf->{'appltype'} = $appltype;#
        $leaf->{'type'} = 'record';
   $leaf->{'view'} = $1;
    }
    elsif (/^:erecord/) {          # test ':erecord'
   $leaf = \%tree;                       # reset $leaf
        $leaf->{'fields'} = $fiecnt;     # save number of fields
        Dump (\%tree);                   # dump information from %element
   %tree = ();                    # clear %element information
   $fiecnt = 0;                   # reset $fiecnt
    }
    elsif (/^:recditem\s.*=\s(\w*)/) {   # test ':recditem'
   $leaf = \%tree;                       # reset $leaf
   $leaf->{"field$fiecnt"}{'name'} = $1;
   $leaf = $leaf->{"field$fiecnt"};
   $fiecnt++;
    }
    elsif (/\s*scope\s*=\s(\w*)/) {      # test 'scope'
        $leaf->{'scope'} = $1;
    }
    elsif (/\s*colname\s*=\s'(\w*)'/) { # test 'colname'
        $leaf->{'colname'} = $1;
    }
    elsif (/\s*readonly\s*=\s(\w)/) {    # test 'readonly'
        $leaf->{'rdonly'} = $1;
    }
    elsif (/\s*key\s*=\s(\w)/) { # test 'key'
        $leaf->{'key'} = $1;
    }
    elsif (/\s*desc\s*=\s'(.*)'/) {      # test 'remark'
        $leaf->{'remark'} = $1;
    }
    elsif (/^:sqltable\s.*=\s'(\w*)'/) {# test ':sqltable'
        $leaf->{'table'} = "\U$1";       # $1 = tablename and '\U' upcase all
    }
}

##
# Close session
#-------------------------------------------------------------------------------
```

A execução do algoritmo do Anexo A produz uma saída do tipo:
(apenas três registros foram listados aqui)

```
found record EC02R12 in line 2499
application                EC02A   type = MAIN
record scope               GLOBAL
table name           RHPESSOAL
number of fields     3
field0 (CDUSUARIO)
   rdonly            Y
   colname           CDUSUARIO
   scope             GLOBAL
   key               Y
field1 (NMPESSOA)
   rdonly            Y
   colname           NMPESSOA
   scope             GLOBAL
   key               N
field2 (CDEMPRESA)
   rdonly            Y
   colname           CDEMPRESA
   scope             GLOBAL
   key               N

found record EC02R09 in line 2551
application                EC02A   type = MAIN
record scope               GLOBAL
table name           IHCADASTRO
number of fields     6
field0 (REGISTRO)
   rdonly            Y
   colname           REGISTRO
   scope             GLOBAL
   key               Y
field1 (NOME)
   rdonly            Y
   colname           NOME
   scope             LOCAL
   key               N
   remark            NOME
field2 (SEXO)
   rdonly            Y
   colname           SEXO
   scope             GLOBAL
   key               N
field3 (DATANASC)
   rdonly            Y
   colname           DATANASC
   scope             GLOBAL
   key               N
field4 (FLOBITO)
   rdonly            Y
   colname           FLOBITO
   scope             GLOBAL
   key               N
field5 (FLATIVO)
   rdonly            Y
   colname           FLATIVO
   scope             GLOBAL
   key               N

found record EC02R08 in line 3019
application                EC02A   type = MAIN
record scope               GLOBAL
table name           HCLOTACAO
number of fields     5
field0 (CDEMPRESA)
   rdonly            Y
   colname           CDEMPRESA
   scope             GLOBAL
   key               Y
field1 (CDLOTACAO)
   rdonly            Y
   colname           CDLOTACAO
   scope             GLOBAL
   key               Y
field2 (DSLOTACAO)
   rdonly            Y
   colname           DSLOTACAO
   scope             GLOBAL
   key               N
field3 (FLATIVO)
   rdonly            Y
   colname           FLATIVO
   scope             GLOBAL
   key               N
field4 (CDSUBORDINACAO06)
   rdonly            Y
   colname           CDSUBORDINACAO06
   scope             GLOBAL
   key               N
```

Anexo B – Algoritmo para extração das relações baseadas em comandos MOVE do código fonte das aplicações

```
##
# Main session
#-------------------------------------------------------------------------------
Command;                            # extract information from command

$leaf = \%tree;                             # get pointer to %tree
while (<>) {                                # hein funny, 'long life to perl'
    $line++;                                # increment line counter
    if (/^:appl\s*name\s*=\s(\w*)/) {   # test ':appl'
   $appl = $1;                      # $1 = application name
   /type\s*=\s(\w*)/;               # get type of application
   $appltype = $1;                          # $1 = application type
    }
    elsif (/^\s*MOVE\s(.*)\sTO\s(.*);/) {
   ##
   # <hipolito, 2002-07-26>
   # in "  MOVE T1.F1 TO T2.F2;"
   # $1 = T1.F1
   # $2 = T2.F2
   # Ps.: Perl is terrific, isn't it?
   #
   $leaf->{'type'} = 'move';        # save type
   $leaf->{'appl'} = $appl;         # get current application
   $leaf->{'appltype'} = $appltype;#
   $leaf->{'line'} = $line;
   $leaf->{'from'}{'move'} = $1;            # save field 'from'
   $leaf->{'to'}{'move'} = $2;              # save field 'to'
   GetFields ($leaf->{'from'}, $1);         # process parts in 'from'
   GetFields ($leaf->{'to'}, $2);                   # process parts in 'to'
   Dump (\%tree);                   # use information in %tree
   %tree = ();                      # reset %tree;
    }
}


##
# Close session
#-------------------------------------------------------------------------------
```

A execução do algoritmo do Anexo B produz uma saída do tipo:

(apenas três comandos foram listados aqui)

```
found MOVE in line 7311
application              EC02A    type = MAIN
EC02R17.NRGPDIAG => EC02R03.NRGPDIAG
   FROM type      tl.fl
   table          EC02R17
   field          NRGPDIAG

   TO type        tl.fl
   table          EC02R03
   field          NRGPDIAG


found MOVE in line 7312
application              EC02A    type = MAIN
EC02R17.NRSGPDIAG => EC02R03.NRSGPDIAG
   FROM type      tl.fl
   table          EC02R17
   field          NRSGPDIAG

   TO type        tl.fl
   table          EC02R03
   field          NRSGPDIAG


found MOVE in line 7313
application              EC02A    type = MAIN
EC02R17.STDIAG => EC02R03.STDIAG
   FROM type      tl.fl
   table          EC02R17
   field          STDIAG

   TO type        tl.fl
   table          EC02R03
   field          STDIAG
```

Anexo C – Algoritmo para extração das relações baseadas em comandos IF do código fonte das aplicações

```
##
# Main session
#-------------------------------------------------------------------------------
Command;                            # extract information from command

$leaf = \%tree;                             # get pointer to %tree
while (<>) {                                # hein funny, 'long life to perl'
    $line++;                                # increment line counter
    if (/^:appl\s*name\s*=\s(\w*)/) {   # test ':appl'
   $appl = $1;                      # $1 = application name
   /type\s*=\s(\w*)/;               # get type of application
   $appltype = $1;                          # $1 = application type
    }
    elsif (/^\s*IF\s(\S*)\s(\S*)\s(\S*);/) {
    ##
    # <hipolito, 2002-07-26>
    # in: "       IF fiel rel fie2;"
    # $1 = fiel
    # $2 = rel
    # $3 = fie2
    #
   $leaf->{'type'} = 'if';                  # save type
   $leaf->{'rel'} = $2;             # save relation
   $leaf->{'appl'} = $appl;         # get current application
   $leaf->{'appltype'} = $appltype;#
   $leaf->{'line'} = $line;
   $leaf->{'from'}{'if'} = $1;              # save field 'from'
   $leaf->{'to'}{'if'} = $3;                # save field 'to'
   GetFields ($leaf->{'from'}, $1);         # process parts in 'from'
   GetFields ($leaf->{'to'}, $3);                   # process parts in 'to'
   if ($leaf->{'to'}{'subtype'} =~ /NRF/) {     # test to PRI
       if (($line - $oldtree{'line'}) < $integrityLine and
            $oldtree{'to'}{'table'} = $leaf->{'from'}{'field'}) {
           ##
           # <hipolito, 2002-07-26>
           # Ok, I found one PRI <possible relational integrity>
           #
           $leaf->{'to'}{'subtype'} = 'f1.PRI';
           $leaf->{'pri'}{'rel'} = $oldtree{'rel'};
           $leaf->{'pri'}{'from'} = $oldtree{'from'};
           $leaf->{'pri'}{'to'} = $oldtree{'to'};
       }
   }
   Dump (\%tree);                   # use information in %tree
   %oldtree = %tree;
   %tree = ();                      # reset %tree;
    }
}


##
# Close session
#-------------------------------------------------------------------------------
```

A execução do algoritmo do Anexo C produz uma saída do tipo:

(apenas três comandos foram listados aqui)

```
found IF statement in line 12619
application              EC03A    type = MAIN
Relation (NE)      EC03R07.CDPROCEDIMENTO => EC03R08.CDPROCEDIMENTO
   FROM type       tl.fl
   table           EC03R07
   field           CDPROCEDIMENTO

   TO type         tl.fl
   table           EC03R08
   field           CDPROCEDIMENTO


found IF statement in line 12624
application              EC03A    type = MAIN
Relation (IS)      EC03R08 => NRF
   TO type         fl.PRI
   table           EC03R08
   POSSIBLE RELATIONAL INTEGRITY :
   Relation (NE)   EC03R07.CDPROCEDIMENTO => EC03R08.CDPROCEDIMENTO

found IF statement in line 12637
application              EC03A    type = MAIN
Relation (EQ)      EC03R07.TPUNIDSOLIC => 'A'
   FROM type       tl.fl
   table           EC03R07
   field           TPUNIDSOLIC
```

Anexo D – Lista das tabelas referenciadas pelo sistema, seus atributos e indicador de quantas vezes ela é referenciada.

TABELA	CAMPO	CHAVE	FREQUENCIA
abagenda	CDAMBULATORIO	yes	1
abagenda	DTCONSULTA	yes	1
abagenda	DTCONSULTAED	yes	1
abagenda	REGISTRO	yes	1
abambulatorio	CDAMBULATORIO	yes	18
abambulatorio	CDLOTACAO	no	2
abambulatorio	DSAMBULATORIO	no	16
abcalendario	ANO	yes	2
abcalendario	DATA	yes	3
abcalendario	DATAPOS	yes	2
abcalendario	DIASEM	yes	3
abcalendario	MES	yes	2
abcalendario	NRDIAS	yes	1
abcalendario	ORDEM	yes	3
abcalendario	DATA	no	1
abcalendario	DIASEM	no	1
abcalendario	DSFERIADO	no	2
abcalendario	FERIADO	no	4
abcalendario	HRHORARIO	no	1
absam	CDSAM	yes	1
absam	DSSAM	no	1
agmov_exame	CDARQEXA	yes	1
agmov_exame	DTEXAME	yes	1
agmov_exame	REGISTRO	yes	1
agmov_exame	CDREQUISITOR	no	1
agmov_exame	DTEXAMEED	no	1
agmov_exame	DTHRMOVIMENTA	no	1
agmov_exame	DTMOVIMENTA	no	1
agmov_exame	HRMOVIMENTA	no	1
agmov_exame	IDREQUISITOR	no	1
agmov_exame	NMMOVIMENTA	no	1
agmov_exame	TPOPERACAO	no	1
ecagenda	CDCONVENIO	yes	15
ecagenda	CDPROCEDIMENTO	yes	15
ecagenda	CDUNIDEXEC	yes	2
ecagenda	DTAGENDA	yes	12
ecagenda	DTAGENDA-AUX	yes	1
ecagenda	DTAGENDAED	yes	2
ecagenda	HRAGENDA	yes	13
ecagenda	NRREGISTRO	yes	16
ecagenda	CDUNIDEXEC	no	12
ecagenda	CDUSUMARCOU	no	9
ecagenda	CDUSUREG	no	1
ecagenda	DTAGENDA	no	3
ecagenda	DTAGENDA1	no	1
ecagenda	DTAGENDAED	no	1
ecagenda	DTHRPRESENTE	no	7
ecagenda	DTHRPRESENTEED	no	1
ecagenda	DTHRREGISTRO	no	1
ecagenda	DTHRREGISTROED	no	1
ecagenda	DTHRREQUISICAO	no	10
ecagenda	FLENCAIXE	no	7
ecagenda	FLEXTRA	no	7
ecagenda	FLLEITO	no	1
ecagenda	FLMOTIVO	no	1
ecagenda	FLPACIENTE	no	11
ecagenda	FLPLANTAO	no	1
ecagenda	FLPRESENCA	no	3
ecagenda	HRAGENDA	no	2
ecagenda	NRRAMAL	no	1
ecagenda	NRREGISTRO	no	2
eccapagenda	CDIDENT	yes	7
eccapagenda	CDTIPO	yes	7
eccapagenda	DTDATA	yes	7
eccapagenda	HRHORARIO	yes	6
eccapagenda	CAPCONVENIOAB	no	7
eccapagenda	CAPCONVENIOIT	no	7
eccapagenda	CAPNORMALAB	no	7

eccapagenda	CAPNORMALIT	no	7
eccapagenda	HRHORARIO	no	1
eccapatend	CDIDENT	yes	10
eccapatend	CDTIPO	yes	10
eccapatend	DIASEM	yes	8
eccapatend	HRHORARIO	yes	10
eccapatend	SEMMES	yes	8
eccapatend	CAPCAB	no	1
eccapatend	CAPCIT	no	1
eccapatend	CAPCONVENIOAB	no	7
eccapatend	CAPCONVENIOIT	no	7
eccapatend	CAPNAB	no	1
eccapatend	CAPNIT	no	1
eccapatend	CAPNORMALAB	no	7
eccapatend	CAPNORMALIT	no	7
eccapatend	CDUSUARIO	no	3
eccapatend	DATA	no	2
eccapatend	DATAED	no	1
eccapatend	DIASEM	no	1
eccapatend	DTHRCONFIRMA	no	3
eccapatend	FERIADO	no	1
eccapatend	FLREDUZ	no	9
eccolreq	CDIDENT	yes	6
eccolreq	CDTIPO	yes	6
eccolreq	DSCOLREQ	yes	3
eccolreq	FLATIVO	yes	3
eccolreq	NRCOLREQ	yes	6
eccolreq	TPCOLREQ	yes	3
eccolreq	DSCOLREQ	no	3
eccolreq	FLATIVO	no	3
eccolreq	TPCOLREQ	no	3